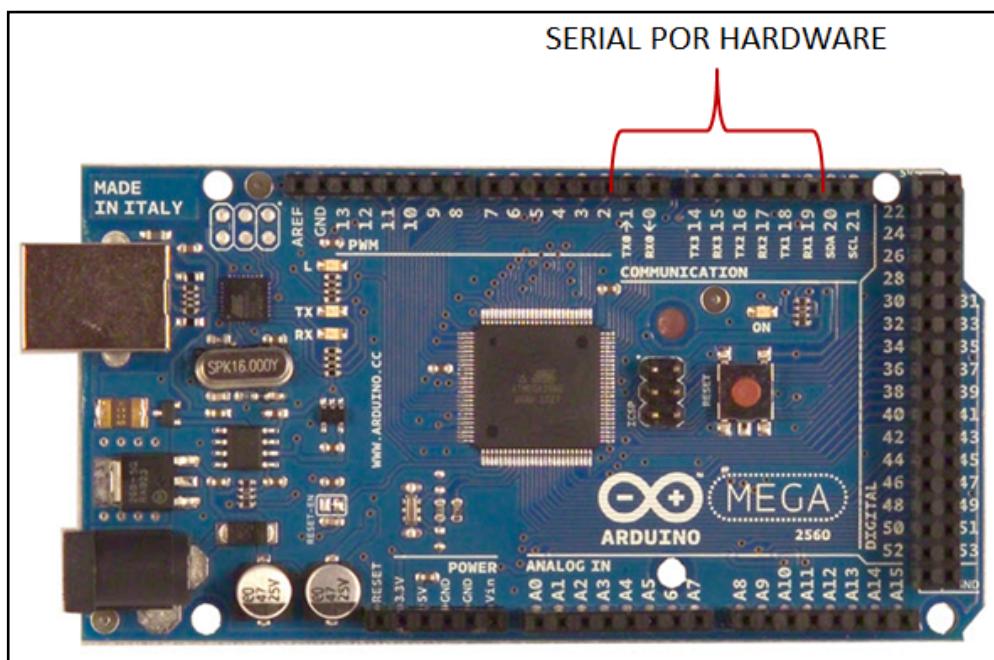
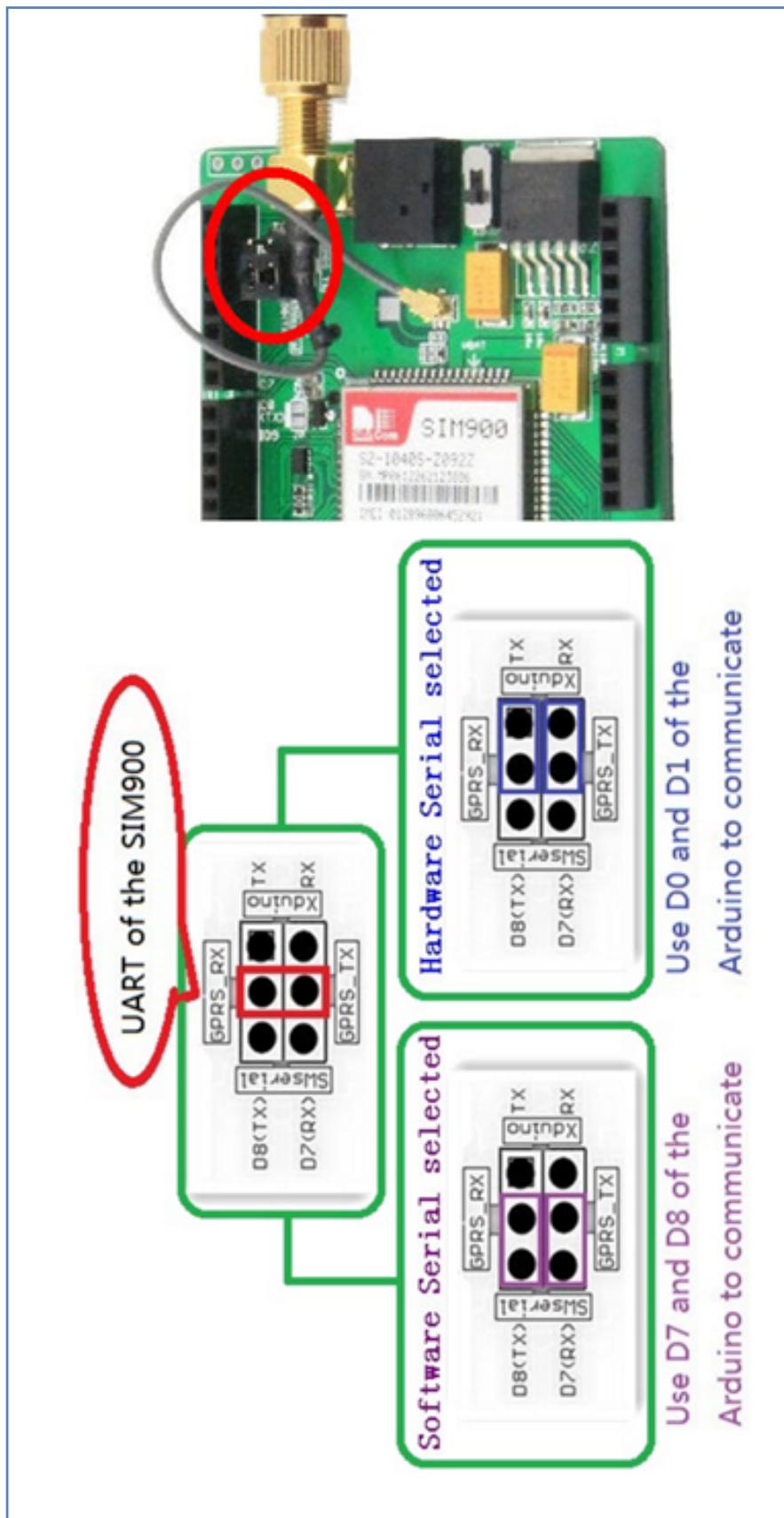


En la siguiente bitácora se dará a conocer las rutinas probadas durante el desarrollo del proyecto **GPRS**. Las rutinas se trataron de forma **aislada** una de otras, con la finalidad, de comprobar el correcto funcionamiento de cada una para luego ser usadas en el proyecto definitivo.

Cabe destacar que, para realizar el control o configuración del módulo **SIM900** se realiza a través de comandos **AT** enviado por el puerto serial del arduino. El módulo **SIM900** tiene la posibilidad de usar el pin D7/D8 como puerto serial por Software, es decir, arduino tiene la posibilidad de invocar una librería que nos permite emular un Serial mediante Software a través del pin D7 y D8, sin embargo, para el Arduino basado en el microcontrolador **MEGA2560** no es posible realizar la comunicación serial a través de estos pines, ya que, no soporta la comunicación a través del pin D7/D8. Es por ello, que se recomienda usar el puerto serial del **Arduino Mega2560** por hardware pin (**0 y 1, 14 y 15, 16 y 17, 18 y 19**).



Para las pruebas de la rutina se usó el puerto Serial por hardware del pin **0 y 1**, para esto se tiene que ajustar el módulo **SIM900** por hardware, el cual, se realiza mediante los **“Jumper”** que se muestra en la siguiente figura.



RUTINA DE ENVÍO DE SMS

Para la ejecución de esta rutina, se realizó la configuración de la **SIM900** mediante comandos **AT** enviados a través del puerto serial (pin 0 y 1).

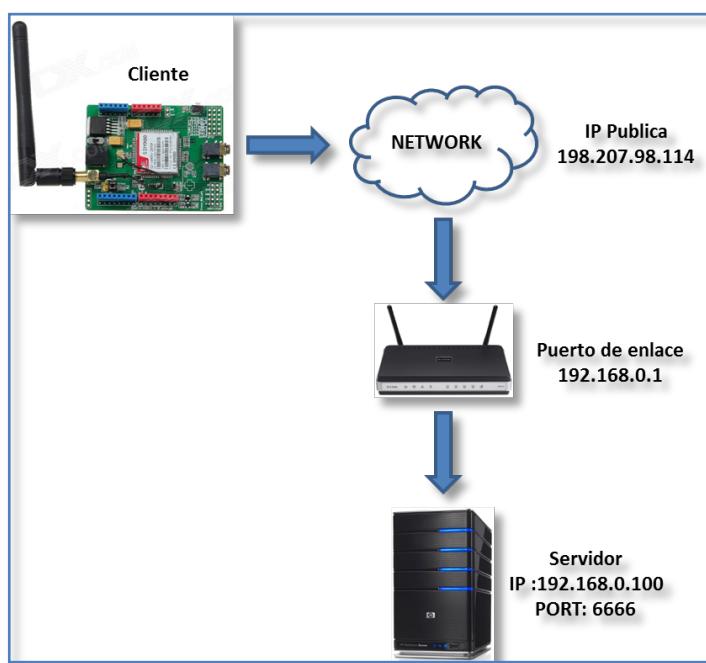
```
1. String numero1="+56953072081"; //Número para enviar el mensaje
2. String mensaje="Bienvenido";/*mensaje a enviar*/ // Mensaje a
   enviar
3. int n=0;
4. String inicio="";
5. void setup()
6. {
7.   Serial.begin(19200); // Se inicia el puerto Serial a 19200
   baudios
8.   delay (5000); //tiempo para ingresar a la red telefónica
9.   Serial.println("AT+CMGF=1\r"); // Configuración del módulo GSM
   en modo texto.
10.   delay(1000);
11.   Serial.println("AT+CNMI=2,2,0,0,0\r"); // el modulo
   enviara los mensaje directamente al puerto serial del
   dispositivo
12.   delay (100);
13.
14. }
15.
16. void loop()
17. {
18.   if (Serial.available ()>0)// Se verifica que el puerto
   Serial esté disponible
19.   //delay (500);
20.   {
21.     inicio=Serial.readStringUntil('\n'); // Se lee el puerto
   Serial cuando entra un mensaje
22.     inicio.trim(); // Se corta los espacio del string leído
23.     if (inicio=="hola")
24.     {
25.       Serial.println("Go");
26.       Serial.println(" AT + CMGS = \"+"+ numero1 +"\""); // Se
   indica el numero el cual el modulo enviara el mensaje mediante
   el comando AT
27.       delay(1000);
28.       Serial.println(mensaje); // se indica el mensaje a enviar
   vía SMS mensaje="Bienvenido"
29.       Serial.println ('\n');// fin de linea
30.       delay(1000);
31.       Serial.write((char)26); // se envía ctrl+z para finalizar
   y enviar el SMS
32.       delay (1000);
33.
34.     }
35.   }
36. }
```

En el encabezado del código se predefine en la variable “**numero1**” el número de teléfono donde se enviara el mensaje de texto y en la variable “**mensaje**” se predefine el SMS que se enviara. Seguidamente, se inicia el puerto serial con 19200 baudios, luego, se envía el comando “**AT+CMGF=1**” para configurar el módulo en modo de recibir y enviar SMS, igualmente, se envía el comando “**AT+CNMI=2,2,0,0,0r**” para indicarle al módulo que recibirá los mensaje directamente por el puerto serial. En el **void loop** básicamente se vigila la llegada de mensaje en el puerto serial del módulo mediante un “**if**” y comparando que el mensaje de llegada sea “**Hola**” para que el modulo responda “**Bienvenido**”.

Se utiliza el comando “**""AT+CMGS=\"""+ numero1 +"\n"** para indicarle al módulo el número de teléfono a donde enviara el **SMS** y luego se envía el mensaje a través de puerto serial. Finalmente, se envía por el puerto serial el carácter “**CTRL+Z**” para finalizar el envío del mensaje.

RUTINA CONEXIÓN TCPIP

Para la rutina de conexión al servidor mediante el protocolo TCPIP se manejaron dos herramientas importantes para llevar a cabo dicha prueba. Primeramente se utilizó un servidor emulado llamado “**HERCULES**” y por último la herramienta “**NO-IP**” para tener una IP fija. Esquema básico de conexión a través de la red.



Cabe destacar que, para la conexión del cliente al servidor se tiene que abrir el puerto a través del enrutador definiendo la **IP** del servidor y el **PORT** (puerto). A continuación rutina de conexión **TCP/IP**.

```
1. int i=0;
2. int pin10=0;
3. String sensor1="";
4. void setup()
5. {
6. pinMode(10,INPUT);
7. Serial.begin(19200);
8. delay(25000); //tiempo para ingresar a la red telefonica
9. }
10.
11.     void loop()
12.     {
13.         if(i==0)
14.             {
15.                 conectgprs(); //Función Conectarse al GPRS
16.             }
17.             enviardato(pin10); // Funcióó enviar dato digital de la
    entrada pin10
18.
19.     }
20.     //FUNCIÓN CONECTAR A
    GPRS///////////
21.     void conectgprs()
22.     {
23.         if (Serial.available ()==0) // Esperar que el serial esté
    disponible
24.         {
25.             if(i==0)
26.             {
27.                 Serial.println("AT+CGATT?\r\n"); // Comando AT para
    entrar a conexión TCP
28.                 delay(5000);
29.                 Serial.println("AT+CSTT=\"web.tmovil.cl\", \"web\", \"web
    \"\r\n"); // Configurar APN, USUARIO y CONTRASEÑA
30.                 delay(5000);
31.                 Serial.println("AT+CIICR\r\n"); // Levantar conexión
    GPRS
32.                 delay(5000);
33.                 Serial.println("AT+CIFSR\r\n"); // Obtener dirección IP
34.                 delay(5000);
35.                 Serial.println("AT+CIPSTART=\"TCP\", \"190.215.92.114\",
    6666\r\n"); // Conectarse al servidor mediante TCP
36.                 delay(5000);
37.                 Serial.println("AT+CIPSEND\r\n"); // Preparar el
    mensaje a enviar
38.                 delay(5000);
39.                 Serial.println("HOLA MUNDO"); // Escribir mensaje a
    enviar
40.                 delay(5000);
41.                 Serial.println("\r\n"); // Entre linea
42.                 Serial.println("\xA"); // Caracter especial para enviar
    mensaje al servidor
43.                 delay(5000);
44.                 i=1;
```

```

45.         }
46.     }
47.   else {
48.     Serial.println("Serial No Disponible");
49.   }
50. }
51. ///////////////////////////////////////////////////////////////////
52.
53. //FUNCION ENVIAR DATO DE LA ENTRADA DIGITAL
54. void enviardato(int x)
55. {
56.   if (Serial.available ()>0)// Esperar datos del
    servidor, es decir, verifica que viene un dato por el puerto
    serial
57.   {
58.     sensor1=Serial.readStringUntil('\n');// Leer petición
    del servidor del sensor 1
59.     sensor1.trim(); // Recortar los espacios
60.     if(sensor1=="sensor") // Comparar petención del sensor
61.     {
62.       Serial.println("AT+CIPSEND\r\n");// Preparar para
    enviar datos de la entrada digital
63.       delay(5000);
64.       x=digitalRead(10); // Leer pin 10 y guardarlo en la
    variable x
65.       Serial.println(x); // Imprimir valor
66.       Serial.println("\r\n");// Entre Linea
67.       Serial.println("\x1A");// Enviar valor de la entrada
    digital pin 10 al servidor
68.       delay(1000);
69.
70.     }
71.   }
72. }

```

Básicamente la rutina de conexión **TCPIP** está basada en comandos **AT**, para ello se realizó una función para conectarse a la red y otra función para enviar dato por petición del servidor. Los comandos **AT** principales utilizados para la conexión son los siguientes.

1. "**AT+CGATT?\r\n**" comando para entrar a conexión **GPRS**.
2. "**AT+CSTT=**"**"web.tmovil.cl"****,****"web"****,****"web"****\r\n**" comando para definir el apn, usuario y contraseña de la red.
3. "**AT+CIICR\r\n**" comando para levantar conexión **GPRS**.
4. "**AT+CIFSR\r\n**" obtener dirección **IP**.
5. "**AT+CIPSTART=**"**"TCP"****,****"190.215.92.114"****,****6666****\r\n**" comando para conectarse al servidor definiendo dirección **IP** y **Puerto** del servidor.
6. "**AT+CIPSEND\r\n**" comando para preparar el modulo para enviar el mensaje al servidor.

7. Se envía el mensaje a través del puerto serial al servidor mediante el código
Serial.println("HOLA MUNDO") // Escribir mensaje a enviar.
8. Se envía el carácter especial “CTRL+Z” o en su defecto “\x1A” con esto se envía el mensaje al servidor.

En el **void loop** del programa, de igual manera, se encuentra la función enviar dato, básicamente esta función espera o compara la petición “**sensor**” del servidor mediante un **IF** para luego enviar el dato requerido, el cual, es una entrada digital del pin 10 del módulo **Arduino**.

RUTINA DE ENVÍOS DE DATOS ANALÓGICOS Y DIGITALES MEDIANTE TCPIP

La rutina que se muestra a continuación está basada en los comandos “**AT**” descrito en la rutina anterior, pero, con envíos de datos de los pin digitales “**10, 11, 12**” y lectura del puerto analógico “**AI0**”. Cabe destacar que, las rutinas de envíos de datos digitales y analógicos se realizaron a través de funciones por paso de parámetros. A continuación se muestra la rutina realizada.

```
1. int i=0;
2. int pin10=10,pin11=11,pin12=12; // Definir las
   entradas digital, 10, 11, 12
3. int ia=0;
4. char opc;
5. String sensor="";
6. void setup()
7. {
8.     for(i=10;i<=12;i++) // For para configurar desde el pin 10 al
   12 como entrada.
9.     {
10.         pinMode (i,INPUT);
11.     }
12.
13.     i=0;
14.     Serial.begin(19200);
15.     delay(25000); //tiempo para ingresar a la red telefónica
16. }
17.
18. void loop()
19. {
20.     if(i==0)
21.     {
22.         conectgprs(); // Función conectar GPRS se ejecuta solo una
   ves al entrar al loop
23.     }
}
```

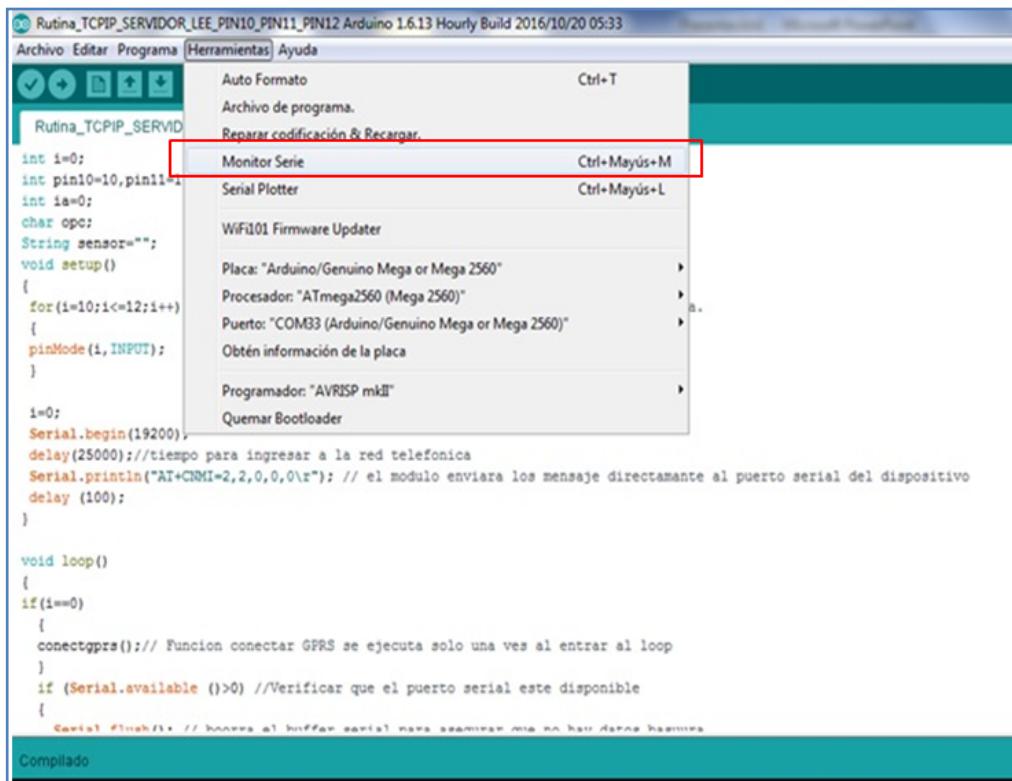
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
24.         if (Serial.available ()>0) //Verificar que viene dato del
    servidor a traves del puerto serial
25.         {
26.             Serial.flush(); // borra el buffer serial para asegurar
    que no hay datos basura
27.             sensor=Serial.readStringUntil ('\n'); // Lee el string
    enviado desde el servidor a traves del puerto serial
28.             sensor.trim(); // Recorta los espacio del string leido
29.             if(sensor=="sensor1")
30.             {
31.                 enviardato(pin10);
32.             }
33.             if(sensor=="sensor2")
34.             {
35.                 enviardato(pin11);
36.             }
37.             if(sensor=="sensor3")
38.             {
39.                 enviardato(pin12);
40.             }
41.             if(sensor=="sensor4")
42.             {
43.                 enviardatoAnalog(ia); // Llama a la función enviar
    dato analogico, AIO
44.             }
45.         }
46.     }
47.     //FUNCIÓN CONECTAR A
    GPRS//////////GPRS//////////GPRS//////////GPRS//////////GPRS
48.     void conectgprs()
49.     {
50.         if (Serial.available ()==0) // Verificar Serial
    Disponible.
51.         {
52.             if(i==0)
53.             {
54.                 Serial.println("AT+CGATT?\r\n"); //Comando AT para
    conectar mediante TCPIP
55.                 delay(5000);
56.                 Serial.println("AT+CSTT=\"web.tmovil.cl\", \"web\", \"web
    \"\r\n"); // Configurar APN, Usuario y Contraseña
57.                 delay(5000);
58.                 Serial.println("AT+CIICR\r\n"); // Levantar señal GPRS
59.                 delay(5000);
60.                 Serial.println("AT+CIFSR\r\n"); //Obtener dirección IP
61.                 delay(5000);
62.                 Serial.println("AT+CIPSTART=\"TCP\", \"190.215.92.114\",
    6666\r\n"); // Conectar al servidor en modo TCP
63.                 delay(5000);
64.                 Serial.println("AT+CIPSEND\r\n"); // Prepara al sim900
    para enviar dato
65.                 delay(5000);
66.                 Serial.println("HOLA MUNDO"); // Escribir dato a enviar
67.                 delay(5000);
68.                 Serial.println("\r\n"); // Entre Linea
69.                 Serial.println("\x1A"); // Enviar dato al servidor, con
    esta linea se indica al sim900 que envie el dato
70.                 delay(5000);
71.                 i=1;
72.             }
```

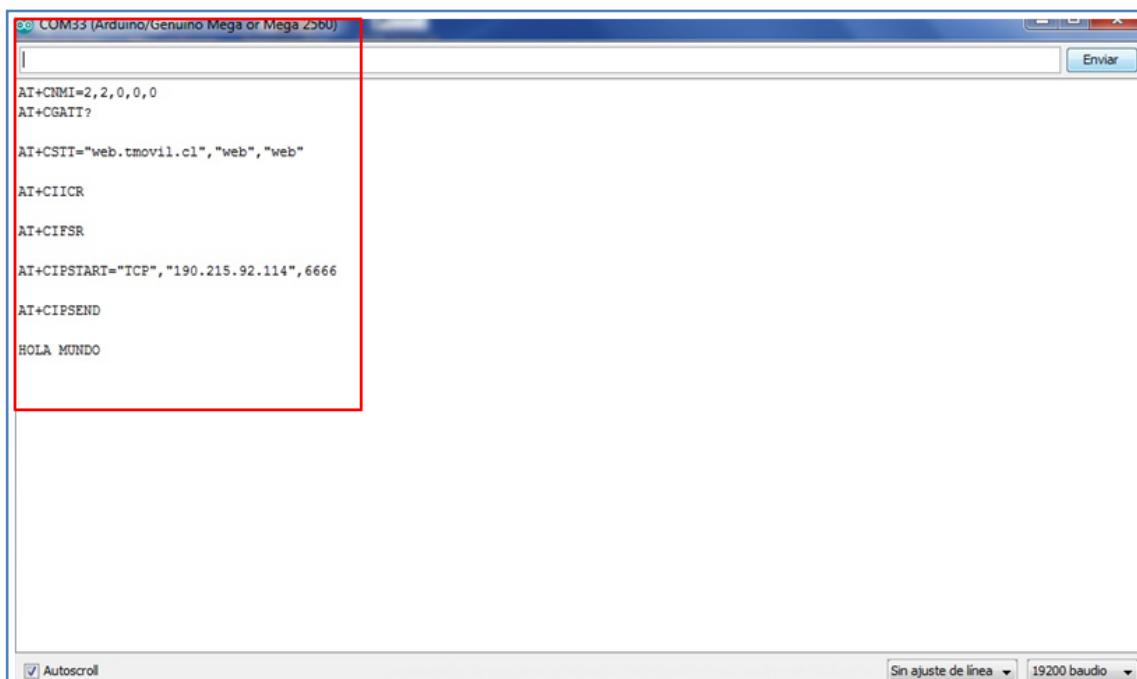
```
73.      }
74.      else {
75.          Serial.println("Serial No Disponible");
76.      }
77.  }
78.  ///////////////////////////////////////////////////
79.  ///////////////////////////////////////////////////
80.  //FUNCION ENVIAR DATO DE LA ENTRADA DIGITAL
81.  void enviardato(int y)
82.  {
83.      int x=0;
84.      n:
85.          Serial.println("AT+CIPSEND\r\n"); // Preparar el modulo
86.          para enviar el dato solicitado
87.          delay(5000);
88.          x=digitalRead(y); // Guardar en la variable x, el pin
89.          solicitado por paso de parametro en la variable y
90.          Serial.println(x);
91.          Serial.println("\r\n"); // Entre linea
92.          Serial.println("\x1A"); // Enviar el dato al servidro con
93.          el caracter especial Ctrl+z
94.          delay(1000);
95.      }
96.  ///////////////////////////////////////////////////
97.  ///////////////////////////////////////////////////
98.  //FUNCION ENVIAR DATO DE LA ENTRADA analogica
99.  void enviardatoAnalog(int y)
100. {
101.     int x=0;
102.     n:
103.     x=analogRead(y); //Se lee el puerto analógico a traves
104.     por paso de parametro de la función en este caso el puerto
105.     analógico 0
106.     Serial.println("AT+CIPSEND\r\n"); // Se prepara el modulo
107.     para enviar el dato del puerto analógico
108.     delay(5000);
109.     Serial.println(x); // Se envia a traves del puerto serial
110.     el dato del puerto a analógico al sim900
111.     Serial.println("\r\n"); // Entre linea
112.     Serial.println("\x1A"); // Se envía al servidor con el
113.     caracter Ctrl+Z
114.     delay(1000);
115. }
116. ///////////////////////////////////////////////////
117. ///////////////////////////////////////////////////
```

En resumen, la rutina se fundamenta en conectarse al servidor a través de **TCPIP** como lo hemos visto en las rutinas anteriores, de igual manera, el envío de datos digitales y datos analógicos se realizan a través de comandos “**AT**” ya antes mencionados. Para comprender un poco más la rutina se mostrara imágenes del envío de comandos “**AT**” del **Arduino** al módulo **SIM900**, para ello, se utiliza el monitor serie del software de programación del **Arduino (Arduino-Nightly)**.

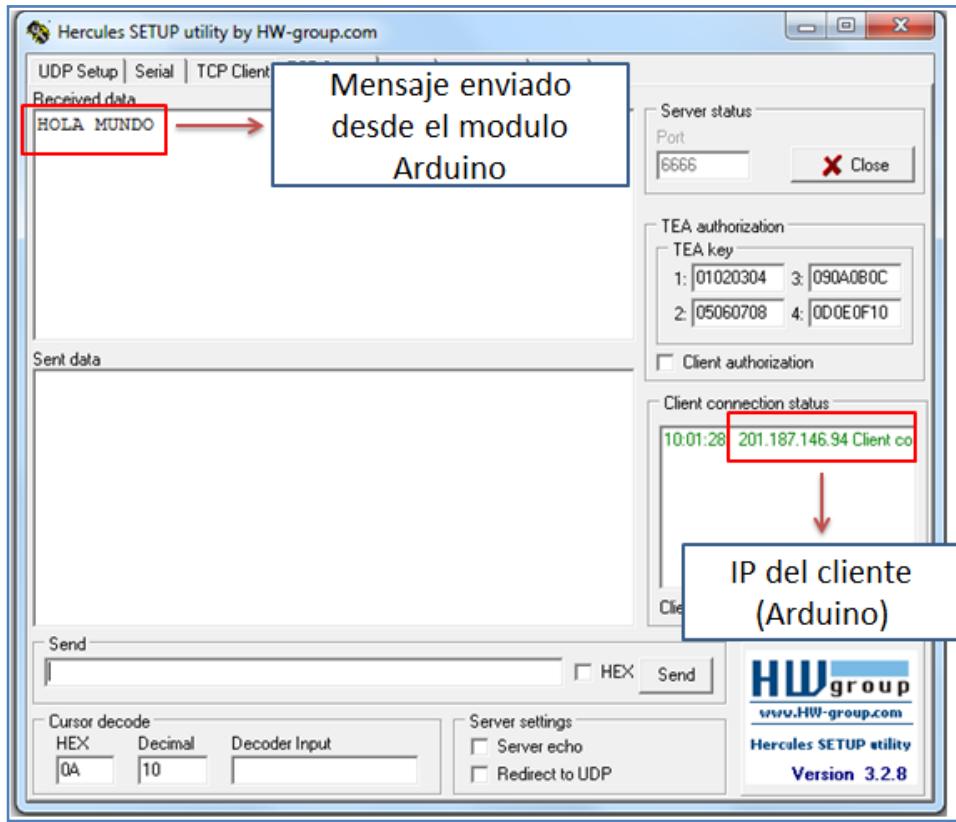
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



A través del monitor serial podemos observar los datos enviados a través del puerto serial del **Arduino**, en la siguiente imagen, veremos los comandos utilizados por la función **conectgprs** a través del monitor serie.

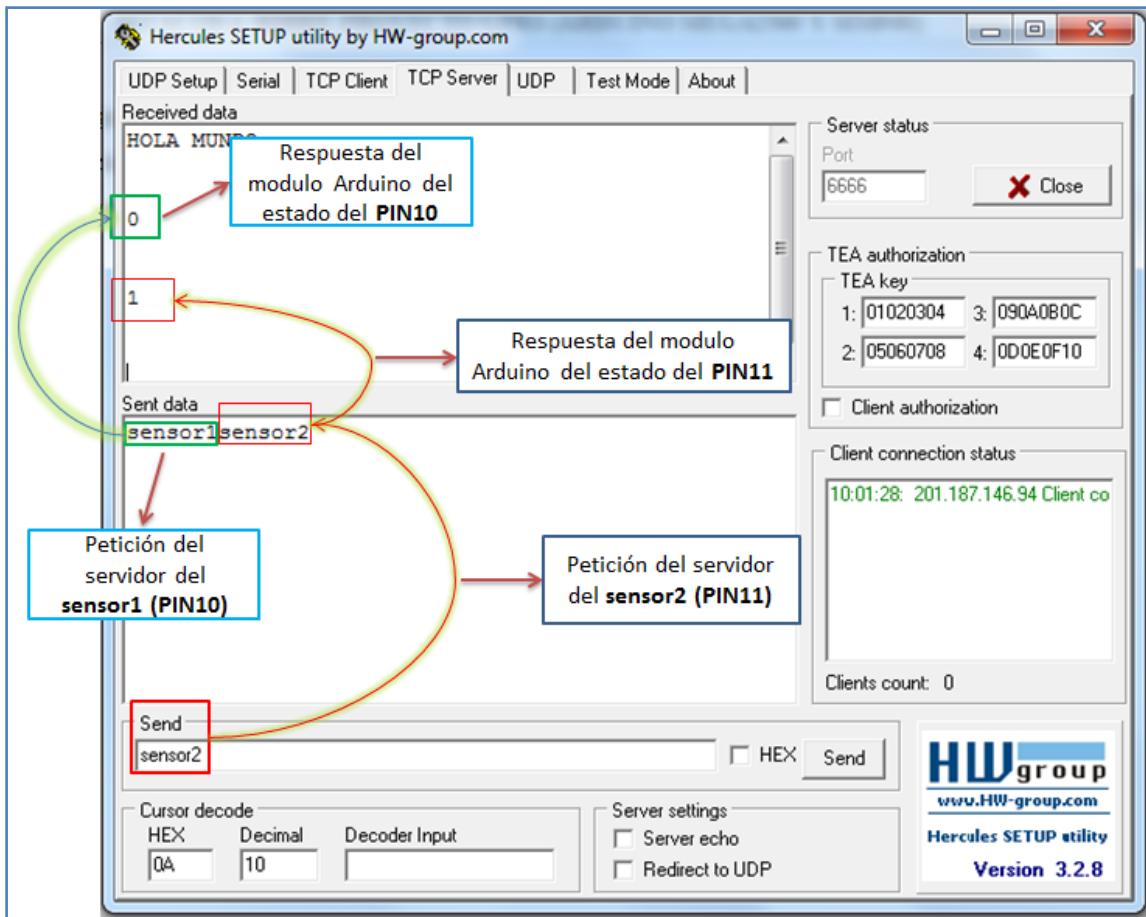


Se observa que, se envía los comandos “AT” para conectarse primeramente al servidor y de la misma forma enviar un mensaje al servidor “**HOLA MUNDO**”.



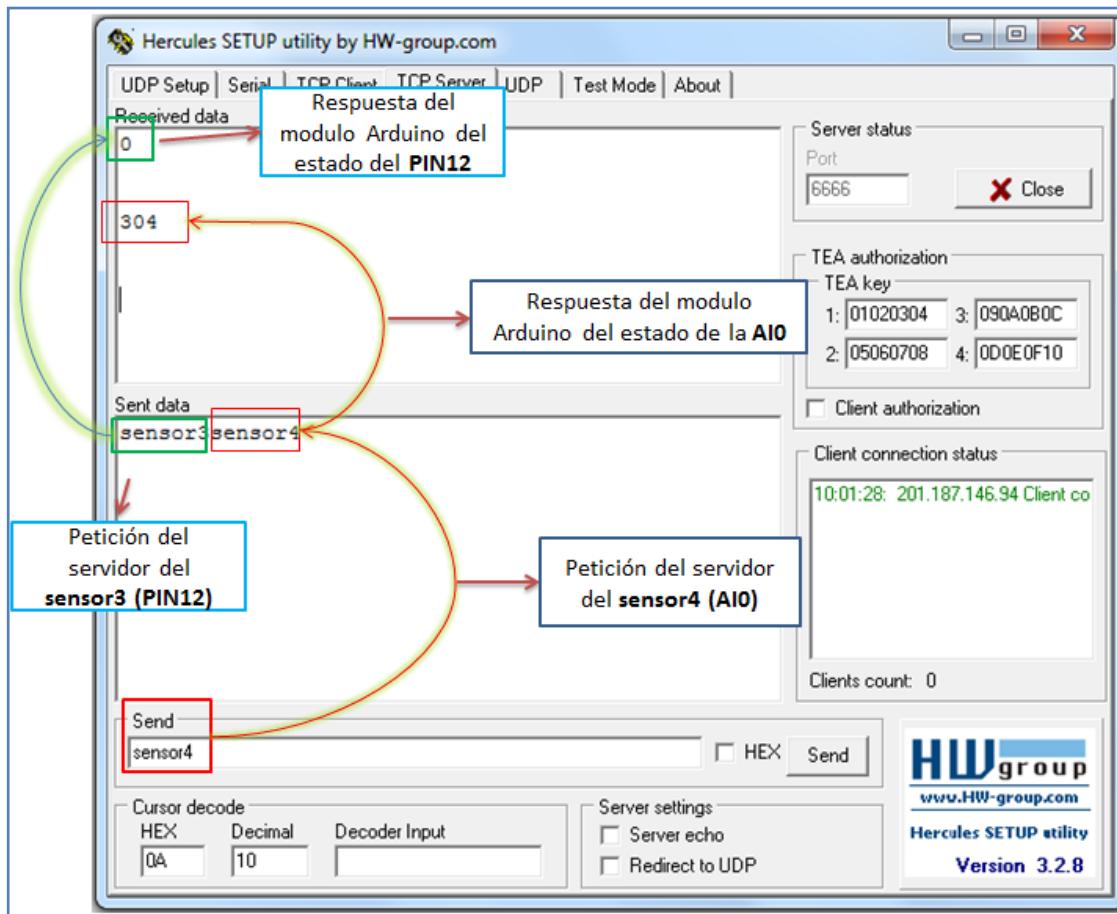
Luego de estar conectado de forma correcta al servidor, el servidor puede hacerle peticiones de los puerto digitales y analógicos antes mencionados de la siguiente manera.

- sensor1 (PIN10)
- sensor2 (PIN11)
- sensor3 (PIN12)
- sensor4 (AI0)



En la imagen se puede apreciar las peticiones realizada por el servidor del **sensor1 (PIN10)** y del **sensor2 (PIN11)**, de igual manera, la respuesta del módulo **Arduino** con el estado de los **PIN10** y **PIN11**. Para el **sensor3** y **sensor4** se realiza de la misma manera tal como se muestra a continuación.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



En la entrada analógica **AI0** se tiene un potenciómetro a **VCC (5v)**, para así variar el voltaje y tener analógicamente el nivel de voltaje en la salida del potenciómetro. Cabe destacar que, la resolución de la entrada analógica es de 10 Bits, por ende, el máximo número decimal es “**1023**” para **5V** y “**0**” para **0V**. En la imagen se aprecia el envío de datos del **sensor3 (PIN12)** y el envío de dato del **sensor4 (AI0)**.

RUTINA DE CONEXIÓN TCPIP CON VERIFICACIÓN DE CONEXIÓN AL SERVIDOR

La rutina de verificación de conexión a servidor se realiza mediante el siguiente comando “**AT+CIPSTATUS**”. Cuando la conexión con servidor es exitosa el módulo **SIM900** responde con el mensaje “**STATE: CONNECT OK**”.

Fundamentalmente la rutina está basada en dos funciones “**connectgprs**” y “**Read**”. La función “**connectgprs**” conecta el modulo al servidor ejecutando todos los comandos explicado anteriormente ([rutina de conexión TCPIP](#)), seguidamente, se tiene la función “**Read**”, dicha función envía el comando “**AT+CIPSTATUS**”, si la respuesta del módulo **SIM900** es positiva entonces la conexión es exitosa, en caso de ser negativa la respuesta, vuelve a reconnectar el módulo al servidor ejecutando la función “**connectgprs**” hasta que la conexión sea exitosa.

Por otro lado, se tiene la función “**clearBuffer**”, dicha función se encarga de limpiar el buffer serial, ya que, cada vez que se envía un comando AT al módulo **SIM900** este responde a través del puerto serial y colapsa de caracteres innecesarios al puerto serial, pues, de este modo el puerto serial queda limpio para leer la respuesta del estatus de conexión del módulo **SIM900** con el servidor. A continuación se muestra la rutina.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
1. ///////////////
2. int i=0,mostrar=0,e=0;
3. String control1,control,control2="STATE: CONNECT OK",Clear; // Se declara la variable control2 para comparar la conexión
4. void setup()
5. {
6.   i=0;
7.   Serial.begin(19200);
8.   delay(25000); //tiempo para ingresar a la red telefónica
9. }
10.
11.   void loop()
12.   {
13.     if(i==0)
14.     {
15.       conectgprs(); // Función conectar GPRS se ejecuta solo
cuando i=0
16.     }
17.     if (i==1)
18.     {
19.       Read(); //Función Read se ejecuta cuando i=1
20.     }
21.   }
22.   ////////////////////FUNCIÓN CONECTAR A
GPRS///////////////////////////////
23.   void conectgprs()
24.   {
25.     if (Serial.available()==0) // Verificar si el serial esta
disponible o vacio
26.     {
27.       Serial.println("AT+CGATT?\r\n"); //Comando AT para
conectar mediante TCPIP
28.       clearBuffer(); // Limpiar Buffer de la respuesta que
envia el SIM900
29.       delay(1000);
30.       Serial.println("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web
\"\r\n"); // Configurar APN, Usuario y Contraseña
31.       clearBuffer();
32.       delay(1000);
33.       Serial.println("AT+CIICR\r\n"); // Levantar señal GPRS
34.       clearBuffer();
35.       delay(1000);
36.       Serial.println("AT+CIFSR\r\n"); //Obtener dirección IP
37.       clearBuffer();
38.       delay(1000);
39.       Serial.println("AT+CIPSTART=\"TCP\",\"190.215.92.114\",
6666\r\n"); // Conectar al servidor en modo TCP
40.       clearBuffer();
41.       delay(1000);
42.       Serial.println("AT+CIPSEND\r\n"); // Prepara sim900
para enviar dato
43.       delay(1000);
44.       Serial.println("HOLA MUNDO"); // Escribir dato a enviar
45.       delay(1000);
46.       Serial.write("\r\n"); // Entre Linea
47.       Serial.write("\x1A"); // Enviar dato al servidor, con
esta linea se indica al sim900 que envie el dato
48.       clearBuffer(); // Limpiamos el buffer
49.       clearBuffer();
50.       //delay(5000);
```

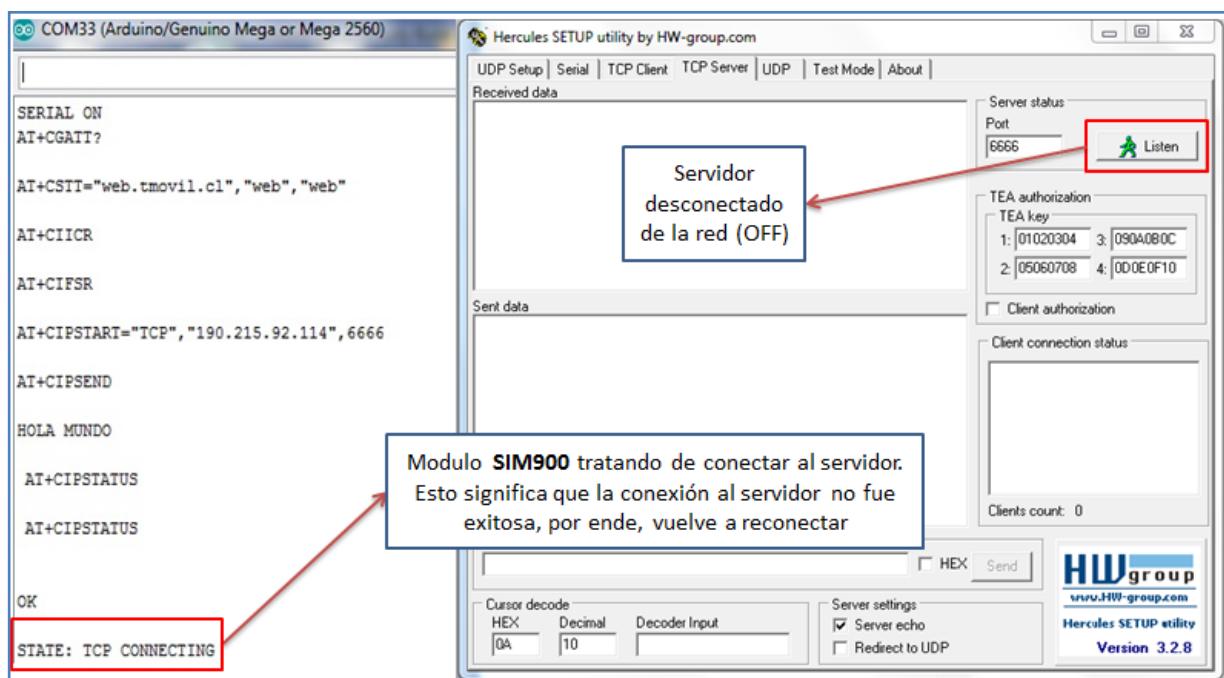
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
51.          i=1;
52.      }
53.  }
54. ///////////////////////////////////////////////////////////////////
55. ///////////////////////////////////////////////////////////////////
56. ///////////////////////////////////////////////////////////////////LIMPIAR
57. void clearBuffer()
58. {
59.
60.     while (Serial.available() !=0)
61.     {
62.         Serial.flush();
63.         Clear=Serial.readStringUntil('\n'); // Limpiando Buffer,
   Guardando respuesta en la variable Clear
64.         Clear=""; // Iniciar variable en 0
65.     }
66. }
67. ///////////////////////////////////////////////////////////////////
68. ///////////////////////////////////////////////////////////////////VERIFICACION DE
CONEXION/////////////////////////////////////////////////////////////////
69. void Read()
70. {
71.     if(Serial.available()==0) // Espera el Serial Disponible
72.     {
73.         Serial.println("AT+CIPSTATUS\r\n"); // Envia el
   comando Status para verificar conexión
74.         delay(100);
75.         while (Serial.available()!=0)// Mientras el buffer
   este lleno de datos se queda en while
76.         {
77.             control=Serial.readStringUntil('\n');// Lee la
   respuesta del modulo SIM900
78.             delay(25);
79.             control1=control; // Pasamos la variable control a
   control1 para luego compararla
80.             control.trim(); // Corta los espacio del string
81.             control1.trim();
82.             Serial.println(control1);
83.         }
84.         if(control1.equals(control2))// Compara la
   respuesta con control2 = "STATE: CONNECT OK"
85.         {
86.             Serial.write("Conexion Exitosa");
87.             i=2;
88.         } else i=0;
89.
90.     }
91.     control=""; // Iniciar la variable en 0 para la proxima
   comparación
92.     control1="";
93. }
94. ///////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
```

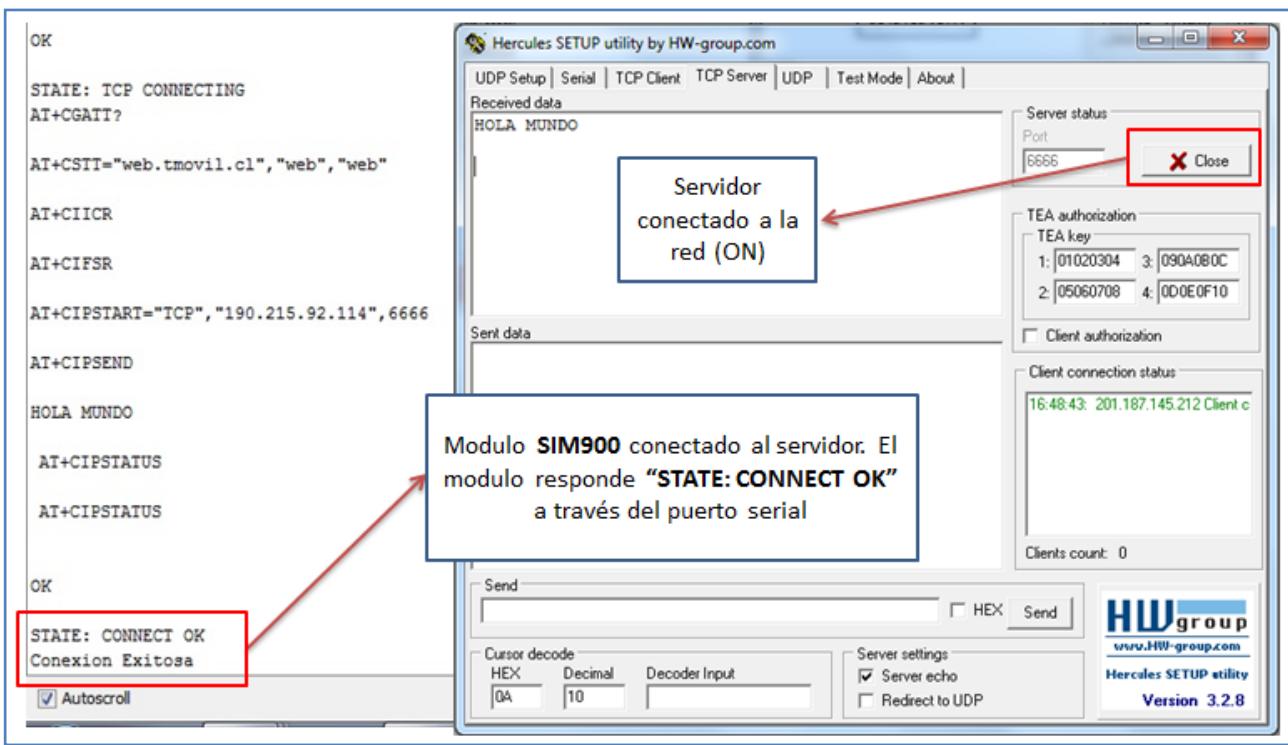
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

En la rutina se puede observar que se usan banderas para controlar el llamado de cada función, es decir, cuando la bandera **i** es igual a “0” se ejecuta la función “**conectgprs**” y si la bandera **i** es igual a “1” se ejecuta la función de comprobar la conexión al servidor. Cabe destacar que, cuando se ejecuta la función “**Read**” y la conexión es **exitosa** la bandera se iguala a “2”, con la finalidad de no volver a ejecutar la función “**conectgprs**” ni la función “**Read**” (función de verificación de conexión).

En las siguientes imágenes, se puede apreciar el caso cuando el servidor está desconectado o apagado y la conexión del módulo **ISM900** con el servidor no es exitosa.



Como se puede observar en la imagen la conexión es fallida, por ende, el programa vuelve a ejecutar la función “**conectgprs**” hasta lograr conectar con el servidor. En la siguiente imagen se observa el módulo **SIM900** conectando de manera efectiva con el servidor.



RUTINAS DE INTERRUPCIÓN INTERNAS

En el siguiente apartado se mostraran detalladamente dos rutinas de interrupción, ya que, con dichas interrupciones podemos verificar la conexión cada cierto tiempo. En primer lugar se detallara la interrupción por desbordamiento del **TIMER4** y luego la interrupción por comparación del **TIMER4**.

Antes de entrar a detallar las rutinas de interrupción explicaremos los registros necesarios para configurar la interrupción y algunos conceptos básicos para entender el funcionamiento del **temporizador (TIMER)**.

- **Timer:** A grandes rasgos, es un contador interno que puede funcionar a la frecuencia que marca un reloj. Este reloj puede ser interno o externo. Este funciona mediante un aumento del “**counter register**”, según como se configure, su contaje será a una frecuencia mayor o menor, y una vez que finalice el contaje (desbordamiento) ya antes configurado activará el bit **flag (bandera)**, el cual, indica que el **TIMER** ha finalizado el contaje y empezará de nuevo. Debido a que

el **Timer** depende de una fuente de reloj en este caso el reloj interno, se usa la siguiente fórmula para calcular el periodo.

$$T = \frac{1}{F} = \frac{1}{16\text{MHz}} = 62,5 \text{ ns}$$

- **Tipos de Timers:** En el **Arduino ATMEGA 2560**, se tiene 6 tipo de **timers**, que se describen a continuación:
 - **Timer 0:** Es un temporizador de 8 bits usado por la función **Delay()** y **millis()**. Cabe destacar que, lo tenemos que tener en cuenta a la hora de programar, ya que, si se está usando las funciones antes mencionadas no se recomienda usar el “Timer 0”.
 - **Timer 1:** Es un temporizador de 16 bits, es usado en la librería de **servo()** (**servomotores**).
 - **Timer 2:** Es un temporizador de 8 bits, similar al “Timer 0” y es usado por la función **tono()**.
 - **Timer 3, 4 y 5:** Estos 3 **Timers** son usados por los **ATMEGA 1280** y **ATMEGA 2560**, son de 16 bits y no están implementados en ninguna función.

Configurar y ejecutar el Timer

Para poder usar los **Timer** o contadores, se deben modificar sus registros. En este caso usaremos el **Timer4**, ya que, no es usado por ninguna función especial del **Arduino**, es por ello, que se modificara los registros **TCCR4A**, **TCCR4B** y **TIMSK4**.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

Los registros tienen la siguiente composición:

- **TCCR4A:** Registro principal del **TIMER4**. Para modificar el modo del **TIMER4**.

| TCCR4A – Timer/Counter 4 Control Register A | | | | | | | | | TCCR4A |
|---|--------|--------|--------|--------|--------|--------|-------|-------|--------|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TCCR4A |
| (0xA0) | COM4A1 | COM4A0 | COM4B1 | COM4B0 | COM4C1 | COM4C0 | WGM41 | WGM40 | |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **TCCR4B:** Registro del **TIMER4**, para modificar el pre-escaler de acuerdo al tiempo de interrupción que se requiera.

| TCCR4B – Timer/Counter 4 Control Register B | | | | | | | | | TCCR4B |
|---|-------|-------|---|-------|-------|------|------|------|--------|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | TCCR4B |
| (0xA1) | ICNC4 | ICES4 | - | WGM43 | WGM42 | CS42 | CS41 | CS40 | |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| CSn2 | CSn1 | CSn0 | Description |
|------|------|------|--|
| 0 | 0 | 0 | No clock source. (Timer/Counter stopped) |
| 0 | 0 | 1 | $\text{clk}_{\text{IO}}/1$ (No prescaling) |
| 0 | 1 | 0 | $\text{clk}_{\text{IO}}/8$ (From prescaler) |
| 0 | 1 | 1 | $\text{clk}_{\text{IO}}/64$ (From prescaler) |
| 1 | 0 | 0 | $\text{clk}_{\text{IO}}/256$ (From prescaler) |
| 1 | 0 | 1 | $\text{clk}_{\text{IO}}/1024$ (From prescaler) |
| 1 | 1 | 0 | External clock source on Tn pin. Clock on falling edge |
| 1 | 1 | 1 | External clock source on Tn pin. Clock on rising edge |

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

- **TIMSK4:** Registro del **TIMER4**, para activar el **TIMER**, a través, del **TOIE4**.

| TIMSK4 – Timer/Counter 4 Interrupt Mask Register | | | | | | | | |
|--|---|---|--------------|---|---------------|---------------|---------------|--------------|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| (0x72) | - | - | ICIE4 | - | OCIE4C | OCIE4B | OCIE4A | TOIE4 |
| ReadWrite | R | R | R/W | R | R/W | R/W | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Primeramente veremos una rutina donde no se activara el pres-escalier del **TIMER4**, es decir, la interrupción se dará por la frecuencia del reloj interno del **Arduino**, por ende, para saber el tiempo de interrupción, se multiplica el periodo **T** ya anteriormente calculado por la resolución (16 Bits) del **TIMER4**. A continuación se muestra el cálculo del tiempo de interrupción.

$$2^{16-1} \times T = 65535 \times 62.5\text{ns} = 0.0041\text{seg} = \mathbf{4.1mseg}$$

Esto quiere decir que, la interrupción se ejecutara cada **4.1mseg**. Cabe destacar que, usaremos la función de interrupción “**ISR (TIMER4_OVF_VECT)**”, esta función interrumpe cuando se desborda el **TIMER4** o cuando la bandera de **OVF (OVERFLOW)** indique un desbordamiento en el **TIMER4**.

| | | |
|------------------------|----------------------|--------------------------------|
| TIMER4_CAPT_vect | SIG_INPUT_CAPTURE4 | Timer/Counter4 Capture Event |
| TIMER4_COMPA_vect | SIG_OUTPUT_COMPARE4A | Timer/Counter4 Compare Match A |
| TIMER4_COMPB_vect | SIG_OUTPUT_COMPARE4B | Timer/Counter4 Compare Match B |
| TIMER4_COMPC_vect | SIG_OUTPUT_COMPARE4C | Timer/Counter4 Compare Match C |
| TIMER4_OVF_vect | SIG_OVERFLOW4 | Timer/Counter4 Overflow |
| TIMER5_CAPT_vect | SIG_INPUT_CAPTURE5 | Timer/Counter5 Capture Event |
| TIMER5_COMPA_vect | SIG_OUTPUT_COMPARE5A | Timer/Counter5 Compare Match A |
| TIMER5_COMPB_vect | SIG_OUTPUT_COMPARE5B | Timer/Counter5 Compare Match B |
| TIMER5_COMPC_vect | SIG_OUTPUT_COMPARE5C | Timer/Counter5 Compare Match C |
| TIMER5_OVF_vect | SIG_OVERFLOW5 | Timer/Counter5 Overflow |

A continuación se muestra la rutina de interrupción de acuerdo a la frecuencia del reloj interna del **ARDUINO**.

```
1. #include <avr/io.h>
2. #include <avr/interrupt.h>
3. int segundos=0;
4. void setup()
5. {
6.     Serial.begin (9600);
7.     cli(); // Deshabilitar interrupciones globales
8.     TCCR4A=0; //Iniciando Registro
9.     TCCR4B=0; //Iniciando Registro
10.    TIMSK4=(1<<TOIE4); // Activando el timer 4
11.    TCCR4B |=(1<<CS40); // Sin prescaler a la frecuencia del
    reloj.
12.    sei(); // Activar las interrupciones
13. }
14.
15. void loop() {
16.
17. }
18.
19. ISR(TIMER4_OVF_vect) // ISR activa las interrupciones del
    TIMER4_OVF_vect, cuando se desborda el contador entra a la
    interrupción ISR
20. {
21.     segundos++;
22.     Serial.println(segundos);
23.     if(segundos==500)
24.     {
25.         Serial.println("Entra");
26.         segundos=0;
27.     }
28. }
```

En esta rutina se aprecia una interrupción a la frecuencia del reloj. Básicamente entra a la interrupción cada **4,1mseg** y aumenta la variable “**segundos**”, para luego, compararla e imprimir el mensaje a través del puerto serial “**Entra**”, de esta forma comprobamos la interrupción.

En el “void setup” de la rutina se tiene la iniciación de los registro del **TIMER4**, seguidamente, se activa el **TIMER4** con el registro **TIMSK4** y sin pre-escalor a través del registro **TCCR4B**. Es importante destacar que, si no se modifica el pre-escalor la interrupción se dará a la frecuencia del reloj y será muy rápida, pues, si se desea tener una interrupción más lenta se debe modificar el pre-escalor en el registro **TCCR4B**.

En este mismo sentido, si deseamos que la interrupción sea cada **1seg**, tendremos que modificar el pre-escalador en el registro **TCCR4B** para así aumentar la resolución del **Timer**. Con la siguiente formula calculamos nuevamente el periodo con un pre-escalador de **1024**.

$$T = \frac{1}{\frac{F}{Pre-escalador}} = \frac{1}{\frac{16MHz}{1024}} = \frac{1024}{16MHz} = 64us$$

Como se puede observar se obtiene un periodo más amplio (**64 Micro segundos**). Seguidamente, se aplica la formula anterior para calcular el tiempo de interrupción del **TIMER4** con un periodo de **65us**.

$$2^{16-1} \times T = 65535 \times 64us = 4.194seg$$

Con el uso de pre-escalador provocamos que el **TIMER** finalice el ciclo cada **4.194seg**, es decir, la interrupción se hará cada **4.194seg**, sin embargo, se requiere que se realice cada **1seg**. Para ello debemos hacer el uso del **CTC**.

El **CTC** se encuentra en el registro **OCR4A**, básicamente, lo que realiza es comparar el valor del **TIMER** con el valor que se le asigne y reinicia el **TIMER** cuando llega al valor asignado para que interrumpa cada **1seg**, es decir, el **TIMER** interrumpe por comparación con el **CTC**, a esta interrupción le llamaremos interrupción por comparación. Para el cálculo del valor de **CTC** se realiza de la siguiente manera.

$$CTC = \frac{\text{Tiempo deseado}}{\text{Periodo del TIMER}} - 1 = \frac{1seg}{64us} - 1 = 15624$$

Este será el valor de **CTC** que le asignara al registro **OCR4**. El **TIMER** se va comparando cada ciclo de reloj con el valor **15624**.

NOTA: EN EL REGISTRO TCCR4B SE ACTIVA EL BIT 4 (WGM42), CON ESTE, INDICAMOS AL TIMER QUE VA A COMPRAR CON EL CTC.

Para la interrupción por comparación, usaremos la siguiente función “**ISR(TIMER4_COMPA_vect)**”.

| | | |
|-------------------|----------------------|--------------------------------|
| TIMER4_CAPT_vect | SIG_INPUT_CAPTURE4 | Timer/Counter4 Capture Event |
| TIMER4_COMPA_vect | SIG_OUTPUT_COMPARE4A | Timer/Counter4 Compare Match A |
| TIMER4_COMPB_vect | SIG_OUTPUT_COMPARE4B | Timer/Counter4 Compare Match B |
| TIMER4_COMPC_vect | SIG_OUTPUT_COMPARE4C | Timer/Counter4 Compare Match C |
| TIMER4_OVF_vect | SIG_OVERFLOW4 | Timer/Counter4 Overflow |

A continuación la rutina con interrupción cada **1seg**.

```

1. #include <avr/io.h>
2. #include <avr/interrupt.h>
3. int segundos=0;
4. void setup()
5. {
6.   Serial.begin (9600);
7.   cli(); // Deshabilitar interrupciones globales
8.   TCCR4A=0; //Iniciando Registro
9.   TCCR4B=0; //Iniciando Registro
10.   OCR4A=15624; // Valor el cual timer se va a comparar con
    el CTC para reiniciar en 1 Seg
11.   TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
12.   TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
    con el CS42
13.   TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
    con el CS40
14.   TIMSK4 |=(1<<OCIE4A); // Activando la comparación
15.   sei(); // Activar las interrupciones
16. }
17.
18. void loop() {
19.
20. }
21.
22. ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
    función cuando la comparación del CTC y el timer es 15624
23. {
24.   segundos++;
25.   Serial.println(segundos);
26.   if(segundos==20) // Entra a la condición luego de 20
    segundos.
27.   {
28.     Serial.println("Entra");
29.     segundos=0;
30.
31.   }
32. }
```

En el “**void setup()**” de la rutina se tiene la iniciación de los registros “**TCCR4A**, **TCCR4B**”, seguidamente, se le asigna el valor **15624** al registro **OCR4A**. Luego, le indicamos al **TIMER4** que la interrupción será por comparación a través del registro **TCCR4B** (línea 11 del código), se activan los pre-escaler (línea 12 y 13 del código) y por último se activa la interrupción por comparación en el registro **TIMSK4** (línea 14 del código).

```
4. void setup()
5. {
6.   Serial.begin (9600);
7.   cli(); // Deshabilitar interrupciones globales
8.   TCCR4A=0; //Iniciando Registro
9.   TCCR4B=0; //Iniciando Registro
10.  OCR4A=15624; // Valor el cual timer se va a comparar con
    el CTC para reiniciar en 1 Seg
11.  TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
12.  TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
    con el CS42
13.  TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
    con el CS40
14.  TIMSK4 |=(1<<OCIE4A); // Activando la comparación
15.  sei(); // Activar las interrupciones
16. }
```

Cabe destacar que, la rutina interrumpe cada **1seg** y se incrementa la variable segundos hasta que sea igual a **20**. Cuando la variable **segundo** es igual a 20, esto indica que ha pasado **20 segundos** y se imprime a través del puerto serial el mensaje “**Entra**”.

NOTA: Con las rutinas de interrupciones, se puede realizar rutinas de control de conexión evaluando la conexión cada cierto tiempo.

RUTINA CON CONTROL DE CONEXIÓN CADA 5 MINUTOS

Para la siguiente rutina se utilizaron códigos de las rutinas anteriormente explicadas, la cuales son; rutina de conexión a servidor, rutina de control de error y rutina de interrupción. En resumen se combinaron las 3 rutinas para así obtener la rutina que mostraremos a continuación.

```
1. /* Notas
2. * Cuando i=0, El programa salta a la función de conectgprs, es
3. * decir, conecta al servidor
4. * Cuando i=1, El programa salta a la función de chequear
5. * estatus de conexión
6. * Cuando i=2, El programa activa la interrupción y coloca la
7. * bandera en i=3, esperando la interrupción para el chequeo de
8. * status
9. */
10.
11. ///////////////////////////////////////////////////////////////////
12. ///////////////////////////////////////////////////////////////////
13. int i=0,e=0,segundos=0;
14. int pin10=10,pin11=11,pin12=12; // Definir las
15. entradas digital, 10, 11, 12
16. int ia=0;
17. String sensor="",control1,control,control2="STATE: CONNECT
18. OK",Clear;
19. ///////////////////////////////////////////////////////////////////
20. void setup()
21. {
22.     for(i=10;i<=12;i++) // For para configurar desde el pin 10
23.     hasta el 12 como entrada.
24.     {
25.         pinMode (i,INPUT);
26.     }
27.     i=0;
28.     Serial.begin(19200);
29.     delay(25000); //tiempo para ingresar a la red telefónica
30.     cli(); // Deshabilitar interrupciones globales
31.     TCCR4A=0; //Iniciando Registro
32.     TCCR4B=0; //Iniciando Registro
33.     OCR4A=15624; // Valor el cual timer se va a comparar con
34.     el CTC para reiniciar en 1 Seg
35.     TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
36.     TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
37.     con el CS42
38.     TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
39.     con el CS40
40.     sei(); // Activar las interrupciones
41. }
42.
43. void loop()
44. {
45.     if(i==0)
46.     {
47.         conectgprs(); // Función conectar GPRS se ejecuta solo una
48.         vez al entrar al loop
```

```
37.      }
38.      if (i==1)
39.      {
40.          Read(); // Función Leer el status de conexión
41.      }
42.      if(i==2)
43.      {
44.          segundos=0;
45.          TIMSK4 =(1<<OCIE4A); // activando interrupción por
comparación
46.          i=3;
47.      }
48.
49.      }
50.      ////////////////////FUNCIÓN CONECTAR A
GPRS///////////////////////////////
51.      void conectgprs()
52.      {
53.          TIMSK4 =(0<<OCIE4A);
54.          if (Serial.available()==0) // Verificar Serial
disponible
55.          {
56.              Serial.println("AT+CGATT?\r\n"); //Comando AT para
conectar mediante TCPIP
57.              clearBuffer(); // Limpiar Buffer de la respuesta que
envia el SIM900
58.              delay(1000);
59.              Serial.println("AT+CSTT=\"web.tmovil.cl\", \"web\", \"web
\"\r\n"); // Configurar APN, Usuario y Contraseña
60.              clearBuffer();
61.              delay(1000);
62.              Serial.println("AT+CIICR\r\n"); // Levantar señal GPRS
63.              clearBuffer();
64.              delay(1000);
65.              Serial.println("AT+CIFSR\r\n"); //Obtener dirección IP
66.              clearBuffer();
67.              delay(1000);
68.              Serial.println("AT+CIPSTART=\"TCP\", \"190.215.92.114\",
6666\r\n"); // Conectar al servidor en modo TCP
69.              clearBuffer();
70.              delay(1000);
71.              Serial.println("AT+CIPSEND\r\n"); // Prepara sim900 para
enviar dato
72.              delay(1000);
73.              Serial.println("HOLA MUNDO"); // Escribir dato a enviar
74.              delay(1000);
75.              Serial.write("\r\n"); // Entre Linea
76.              Serial.write("\x1A"); // Enviar dato al servidor, con
esta linea se indica al sim900 que envie el dato
77.              clearBuffer();
78.              clearBuffer();
79.              i=1;
80.          }
81.      }
82.      /////////////////////////////////
83.      //////////////////LIMPIAR
BUFFER/////////////////////////////
84.      void clearBuffer()
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
86.      {
87.
88.      while (Serial.available() !=0)
89.      {
90.          Serial.flush();
91.          Clear=Serial.readStringUntil('\n'); // Limpiando Buffer,
   Guardando respuesta en la variable Clear
92.          Clear=""; // Iniciar variable en 0
93.      }
94.  }
95.  /////////////////////////////////
// 
96.  ///////////////////VERIFICACION DE
CONEXION/////////////////////////////
97.  void Read()
98.  {
99.      if(Serial.available()==0) // Espera el Serial
Disponible
100.     {
101.         TIMSK4 =(0<<OCIE4A);
102.         Serial.println("AT+CIPSTATUS\r\n"); // Envia el comando
Status para verificar conexión
103.         delay(100);
104.         while(Serial.available()!=0)// Mientras el buffer
este lleno de datos se queda en while
105.         {
106.             control=Serial.readStringUntil('\n');// Lee la
respuesta del modulo SIM900
107.             delay(25);
108.             control1=control;
109.             control.trim(); // Corta los espacio del string
110.             control1.trim();
111.         }
112.         if(control1.equals(control2))// Compara la
respuesta con control2 = "STATE: CONNECT OK"
113.         {
114.             Serial.println("Conexion Exitosa");
115.             i=2;
116.         }else
117.         {
118.             Serial.println("Conexion Fallida");
119.             i=0;
120.         }
121.
122.     }
123.     clearBuffer(); // Limpiar buffer
124.     control1=""; // Limpiar Variables
125.     control=""; // Limpiar Variables
126.   }
127.   /////////////////////////////////
// 
128.   //////////////////Función Interrumpir para verificar
Status///////////////////
129.   ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
función cuando la comparación del CTC y el timer es 15624
130.   {
131.       segundos++;
132.       if(segundos==30) // Entra a la condición luego de 5
minutos.
133.   }
```

```
134.      i=1;
135.      segundos=0;
136.    }
137.  }
138. ///////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////
```

En el “**void setup**” principalmente tenemos la inicialización de los registro de interrupción, de igual manera, se tiene la configuración de los pines como entrada, pero, en esta rutina no se utilizaran dichas entradas configuradas.

En el “**void loop**” tenemos 3 funciones, la cuales, se ejecutaran dependiendo de la bandera “**i**”. Las funciones se ejecutaran como se explica a continuación:

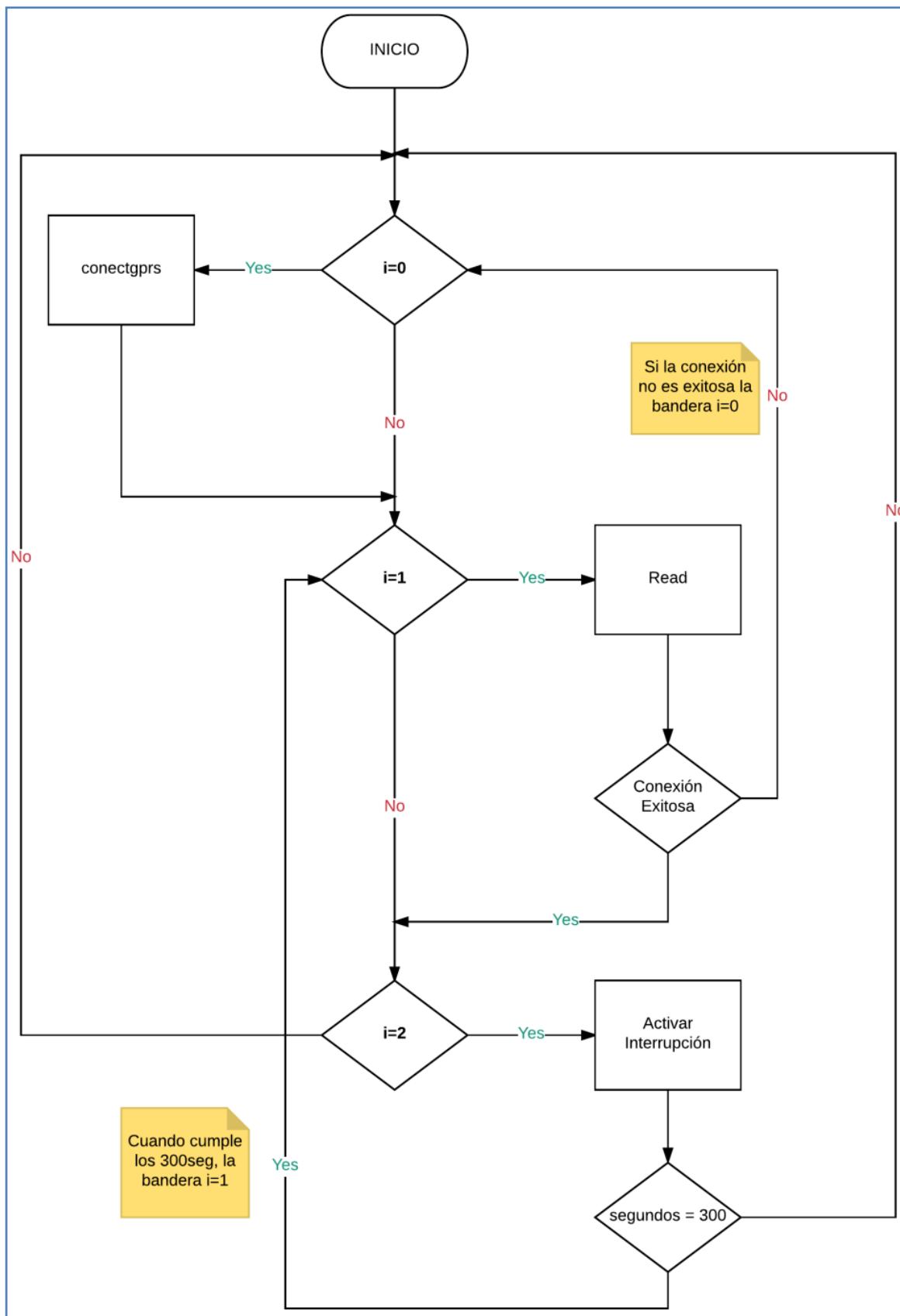
- **Cuando “i=0”:** Se ejecuta la función **conectgprs**, esta función se ejecuta al inicio del programa para conectar el modulo al servidor. Dentro la misma función se coloca la bandera “**i=1**” para luego verificar el estatus de conexión con la función **Read**. La función **conectgprs**, esta explicada con más detalles en la rutina de [conexión TCPIP](#). Sin embargo, se le agregaron algunas líneas de código, pero, sin modificar el código base.
- **Cuando “i=1”:** Se ejecuta la función **Read**, esta función verifica el estatus de conexión con el servidor mediante comandos **AT** a través del puerto serial. Cuando la conexión es exitosa, coloca la bandera “**i=2**” para activar la interrupción y verificar la conexión cada 5 minutos, cuando la conexión no es exitosa, coloca la bandera “**i=0**” para volver a ejecutar la función **conectgprs**. La función **Read**, esta explicada con más detalles en la rutina de [conexión TCPIP con verificación de conexión al servidor](#). Sin embargo, se le agregaron algunas líneas de código, pero, sin modificar el código base.
- **Cuando “i=2”:** Se activa la interrupción por comparación y coloca la bandera “**i=3**”, para que así, el programa quede solo en el “**void loop**” y no entre a ninguna condición. La interrupción se realiza cada **1seg** e incrementa la variable **segundos** y cuando esta llegue a **300** (300seg = 5Min) se coloca la bandera “**i=1**” para verificar el estatus de conexión al servidor mediante la función **Read**.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
134.     ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
    función cuando la comparación del CTC y el timer es 15624
135.     {
136.         segundos++;
137.         if(segundos==300) // Entra a la condición luego de 5
    minutos.
138.         {
139.             i=1;
140.             segundos=0;
141.         }
142.     }
```

A continuación se muestra un diagrama de flujo para entender un poco mejor la rutina.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



RUTINA CON CONTROL DE CONEXIÓN CADA 60 SEGUNDOS Y PETICIÓN DEL SERVIDOR

Para llevar a cabo esta rutina se utilizaron las siguientes rutinas ya expuesta anteriormente:

- Rutina de [conexión TCPIP](#).
- Rutina de [envíos de datos analógicos y digitales mediante TCPIP](#).
- Rutina [conexión TCPIP con verificación de conexión al servidor](#).
- Rutina con [control de conexión cada 5 minutos](#).

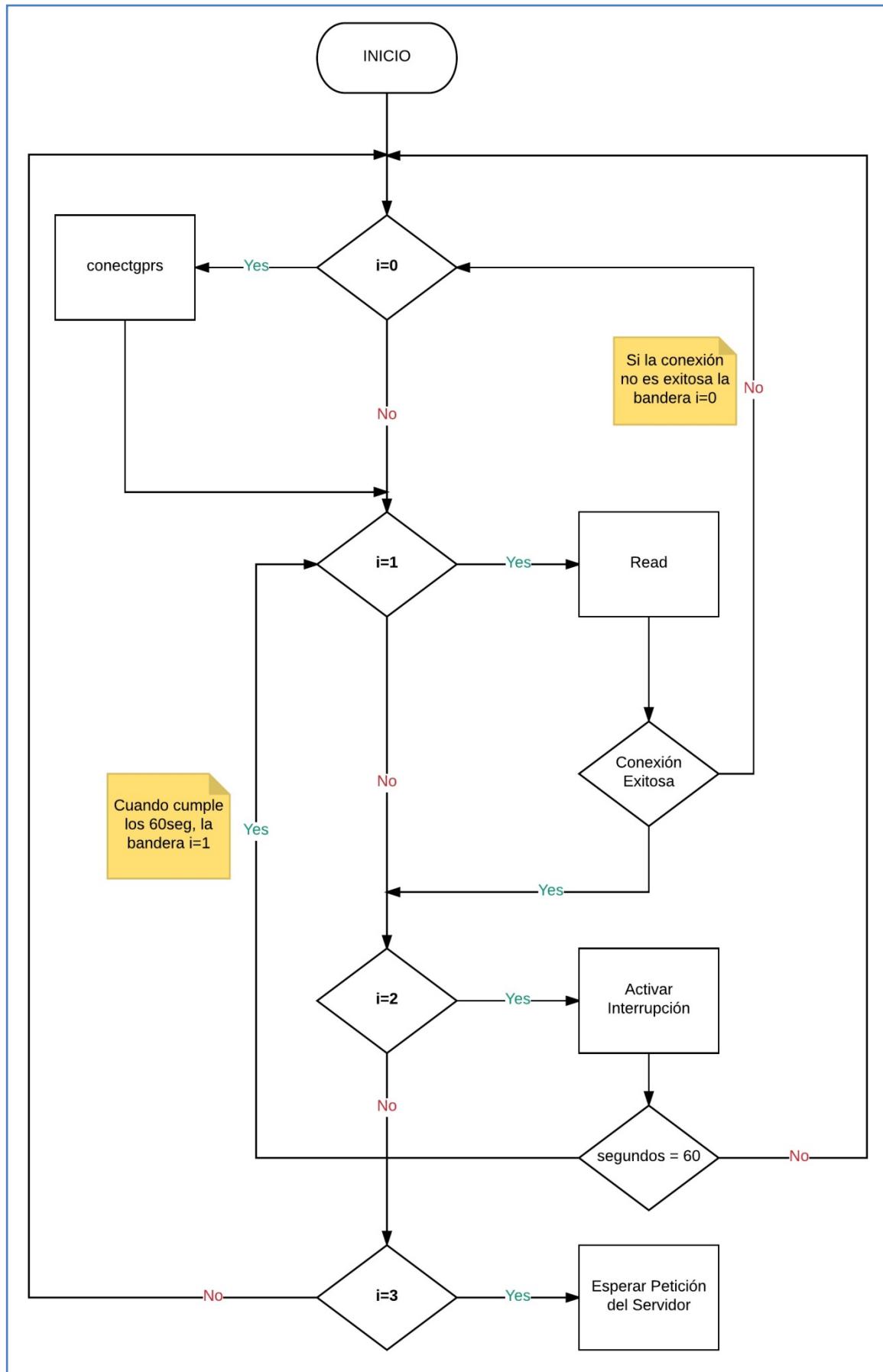
Cabe destacar que, se agregaron algunas líneas de código sin alterar el código base de las rutinas, ya que, se realizó un ensamble de todas las rutinas antes mencionadas.

En el “**void loop**” se agregó una condición de la bandera “**i=3**”, para dejar el programa dentro de un “**while**” y así esperar la petición del servidor. Luego de pasar los 60 segundos, se coloca la bandera “**i=1**” para verificar el estatus de conexión y entrar en las condiciones dictadas por la bandera **i**.

De igual manera, se agregaron las funciones de enviar datos digitales y analógicos a través de paso de parámetro, dichas funciones están detalladas en la rutina de [envíos de datos analógicos y digitales mediante TCPIP](#).

A continuación se presenta un diagrama de flujo de las condiciones medulares de la rutina:

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



A continuación se muestra el código de la rutina de control de conexión y petición del servidor:

```
1. /* Notas
2. * Cuando i=0, El programa salta a la función de conectgprs, es
3. * decir, conecta al servidor
4. * Cuando i=1, El programa salta a la función de chequear
5. * estatus de conexión
6. * Cuando i=2, El programa activa la interrupción y coloca la
7. * bandera en i=3, esperando la interrupción para el chequeo de
8. * status
9. * Cuando i=3, El programa se queda en el while esperando
10. * petición del servidor
11. */
12. ///////////////////////////////////////////////////
13. int i=0,e=0,segundos=0;
14. int pin10=10,pin11=11,pin12=12; // Definir las
15. entradas digital, 10, 11, 12
16. int ia=0;// Entrada analógica
17. String sensor="",control1,control,control2="STATE: CONNECT
18. OK",Clear;
19. ///////////////////////////////////////////////////
20. void setup()
21. {
22.     for(i=10;i<=12;i++) // For para configurar desde el pin 10
23.         hasta el 12 como entrada.
24.     {
25.         pinMode(i,INPUT);
26.     }
27.     i=0;
28.     Serial.begin(19200);
29.     delay(25000); //tiempo para ingresar a la red telefónica
30.     cli(); // Deshabilitar interrupciones globales
31.     TCCR4A=0; //Iniciando Registro
32.     TCCR4B=0; //Iniciando Registro
33.     OCR4A=15624; // Valor el cual timer se va a comparar con
34.         el CTC para reiniciar en 1 Seg
35.     TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
36.     TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
37.         con el CS42
38.     TCCR4B |=(1<<CS42); //Activar el kprescaler a 1024
39.         conjunto con el CS40
40.     sei(); // Activar las interrupciones
41. }
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
42.      Read(); // Función Leer el status de conexión
43.      }
44.      if(i==2)
45.      {
46.          segundos=0;
47.          TIMSK4 = (1<<OCIE4A); // activando interrupción por
comparación
48.          i=3;
49.      }
50.
51.      if(i==3)
52.      {
53.          if(Serial.available ()>0) //Verificar que hay dato en el
puerto serial, o que viene dato desde el servidor.
54.          {
55.              Serial.flush(); // borra el buffer serial para asegurar
que no hay datos basura
56.              sensor=Serial.readStringUntil('\n'); // Lee el string
enviado desde el servidor a traves del puerto serial
57.              sensor.trim(); // Recorta los espacio del string leido
58.              if(sensor=="sensor1")
59.              {
60.                  enviardato(pin10);
61.              }
62.              if(sensor=="sensor2")
63.              {
64.                  enviardato(pin11);
65.              }
66.              if(sensor=="sensor3")
67.              {
68.                  enviardato(pin12);
69.              }
70.              if(sensor=="sensor4")
71.              {
72.                  enviardatoAnalog(ia); // Llama a la función enviar
dato analógico, AIO
73.              }
74.
75.          }
76.      }
77.  }
78.
79. ////////////////FUNCIÓN CONECTAR A
GPRS///////////////
80. void conectgprs()
81. {
82.     if (Serial.available()==0) // Espera que el serial este
disponible
83.     {
84.         Serial.println("AT+CGATT?\r\n"); //Comando AT para
conectar mediante TCPIP
85.         clearBuffer(); // Limpiar Buffer de la respuesta que
envia el SIM900
86.         delay(1000);
87.         Serial.println("AT+CSTT=\"web.tmovil.cl\", \"web\", \"web
\"\r\n"); // Configurar APN, Usuario y Contraseña
88.         clearBuffer();
89.         delay(1000);
90.         Serial.println("AT+CIICR\r\n"); // Levantar señal GPRS
91.         clearBuffer();
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
92.         delay(1000);
93.         Serial.println("AT+CIFSR\r\n"); //Obtener dirección IP
94.         clearBuffer();
95.         delay(1000);
96.         Serial.println("AT+CIPSTART=\"TCP\", \"190.215.92.114\",
  6666\r\n"); // Conectar al servidor en modo TCP
97.         clearBuffer();
98.         delay(1000);
99.         Serial.println("AT+CIPSEND\r\n"); // Prepara sim900
  para enviar dato
100.        delay(1000);
101.        Serial.println("HOLA MUNDO"); // Escribir dato a enviar
102.        delay(1000);
103.        Serial.write("\r\n"); // Entre Linea
104.        Serial.write("\x1A"); // Enviar dato al servidor, con
  esta linea se indica al sim900 que envie el dato
105.        clearBuffer();
106.        clearBuffer();
107.        i=1;
108.    }
109.}
110.///////////
111./////////
112.//////////LIMPIAR
  BUFFER///////////
113.    void clearBuffer()
114.    {
115.
116.        while (Serial.available() !=0)
117.        {
118.            Serial.flush();
119.            Clear=Serial.readStringUntil('\n'); // Limpiando Buffer,
  Guardando respuesta en la variable Clear
120.            Clear=""; // Iniciar variable en 0
121.        }
122.    }
123.///////////
124.//////////VERIFICACION DE
  CONEXION///////////
125.    void Read()
126.    {
127.        if(Serial.available()==0) // Espera el Serial Disponible
128.        {
129.            TIMSK4 = (0<<OCIE4A);
130.            Serial.println("AT+CIPSTATUS\r\n"); // Envia el
  comando Status para verificar conexión
131.            delay(100);
132.            while(Serial.available()!=0)// Mientras el buffer
  este lleno de datos se queda en while
133.            {
134.                control=Serial.readStringUntil('\n');// Lee la
  respuesta del modulo SIM900
135.                delay(25);
136.                control1=control;
137.                control.trim(); // Corta los espacio del string
  control1.trim();
138.            }
139.        }
140.
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
141.             if(control1.equals(control2)) // Compara la
    respuesta con control2 = "STATE: CONNECT OK"
142.                 {
143.                     Serial.println("Conexion Exitosa");
144.                     i=2;
145.                 }else
146.                 {
147.                     Serial.println("Conexion Fallida");
148.                     i=0;
149.                 }
150.             }
151.             clearBuffer();
152.             control="";
153.             control1="";
154.         }
155.         /////////////////
156.         //Funcion Interrumpir para verificar
    Status///////////
157.     ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
    función cuando la comparación del CTC y el timer es 15624
158.     {
159.         segundos++;
160.         if(segundos==60) // Entra a la condición luego de 5
    minutos.
161.         {
162.             i=1;
163.             segundos=0;
164.         }
165.     }
166.     /////////////////
167.     //FUCNION ENVIAR DATO DE LA ENTRADA DIGITAL
    /////////////
168.     void enviardato(int y)
169.     {
170.         int x=0;
171.         Serial.println("AT+CIPSEND\r\n"); // Preparar el modulo
    para enviar el dato solicitado
172.         delay(1000);
173.         x=digitalRead(y); // Guardar en la variable x, el pin
    solicitado por paso de parametro en la variable y
174.         Serial.println(x);
175.         Serial.println("\r\n"); // Entre linea
176.         Serial.println("\x1A"); // Enviar el dato al servidor con
    el caracter especial Ctrl+z
177.         delay(1000);
178.     }
179.     /////////////////
180.     //FUNCION ENVIAR DATO DE LA ENTRADA ANALOGICA
    /////////////
181.     void enviardatoAnalog(int y)
182.     {
183.         int x=0;
184.         x=analogRead(y); //Se lee el puerto analógico a traves
    por paso de parametro de la función en este caso el puerto
    analógico 0
185.         Serial.println("AT+CIPSEND\r\n"); // Se prepara el modulo
    para enviar el dato del puerto analógico
```

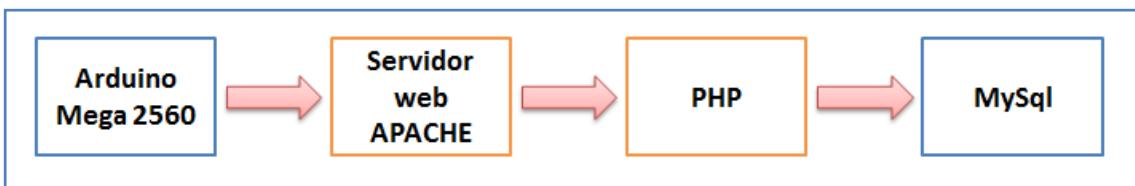
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
186.     delay(1000);
187.     Serial.println(x); // Se envia a traves del puerto serial
    el dato del puerto a analógico al sim900
188.     Serial.println("\r\n"); // Entre linea
189.     Serial.println("\x1A"); // Se envia al servidor con el
    caracter Ctrl+Z
190.     delay(1000);
191.
192. }
193. /////////////////
//////
```

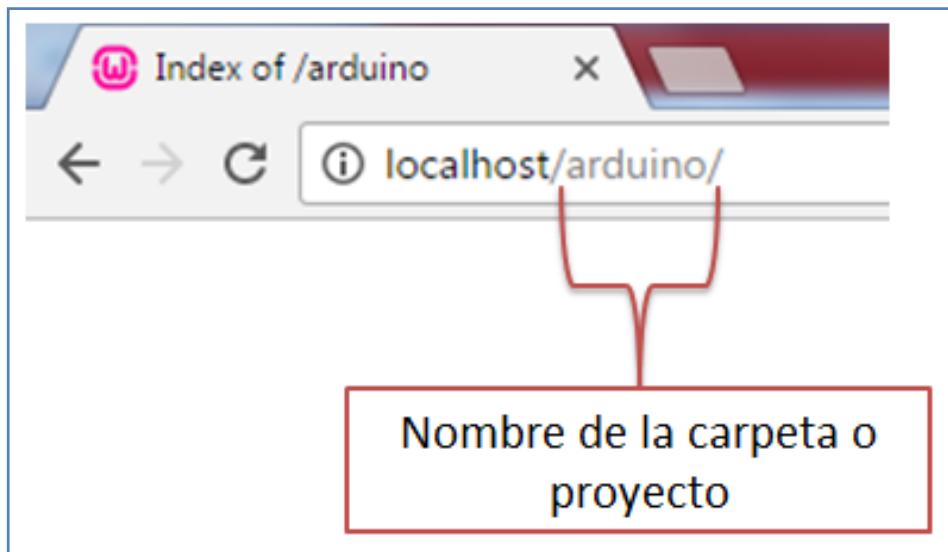
CONEXION A BASE DE DATOS MEDIANTE PHP

Con las rutinas descritas anteriormente es posible enviar datos y recibir peticiones de un servidor simulado, sin embargo, no se puede guardar estos datos en una base de datos, por ende, se realizaron rutinas basadas en las rutinas anteriores para enviar datos y guardarlos en una base de datos.

Para ello, se instaló **wampserver**, ya que, este tiene integrado el **Servidor Web (APACHE)**, **PHP** y el motor de base de datos **Mysql**. Cabe destacar que, para introducir los valores en la base de datos **Mysql** se realiza mediante códigos **PHP**, ya que, el **arduino** no posee una librería propia que permita conectarse directamente al motor de base de datos. Es decir, el **arduino** enviará los datos al servidor web **Apache**, este tomará los datos y los enviará a **Mysql**, la programación de colección de datos o la captura de datos será bajo **PHP**. A continuación se muestra un pequeño diagrama de conexión para visualizar la conexión de arduino a través de **PHP**.



Ahora bien, **wampserver** nos permite programar en **PHP** como se ha mencionado anteriormente. Las programaciones en **PHP** se realizan a través de editores de texto (Bloc de notas, Notepad++, etc.), para ejecutar estos códigos se debe crear una carpeta con el nombre del proyecto donde se guardaran los códigos **PHP**, la ruta de la carpeta donde se guardaran los proyectos es la siguiente **C: //wampserver64/www**. Es importante destacar que, para abrir o ejecutar los proyectos guardados en la ruta anteriormente mencionada, se realiza a través del navegador web escribiendo en la barra de navegación el nombre del proyecto o carpeta de la siguiente manera:

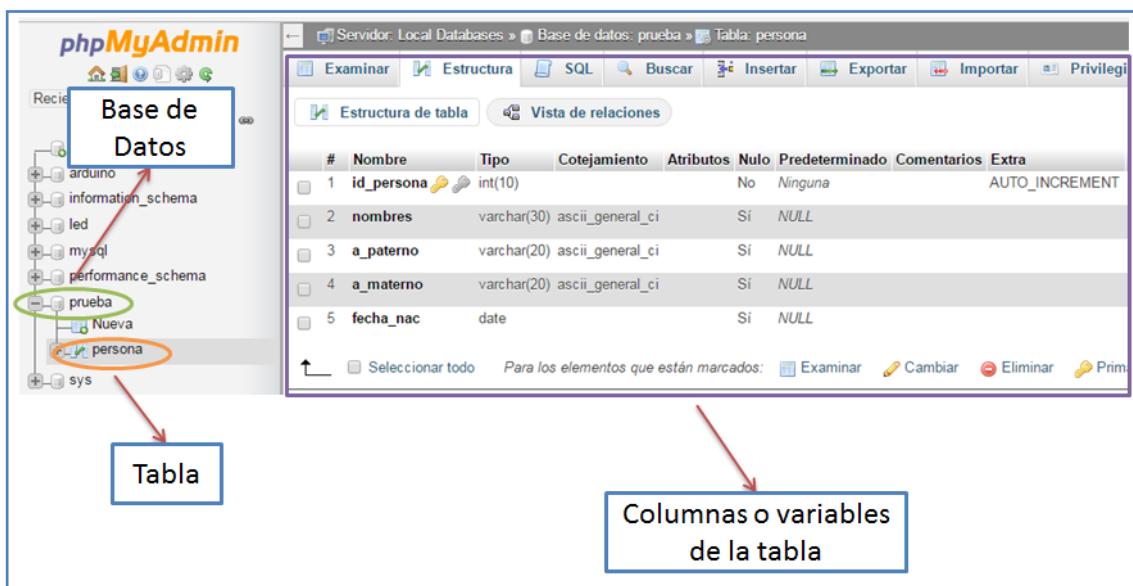


Cabe destacar que, para ejecutar el proyecto desde el navegador se tiene que iniciar los servicios de **Wampserver** o simplemente ejecutar **Wampserver**.

Los códigos en **PHP** se realizan separados dependiendo de la necesidad del proyecto a ejecutar, es decir, en un código se puede tener el llenado del formulario, en otro código la conexión de a **Mysql** y en otro código la recolección de datos de ese formulario para introducirlo en la base de datos.

Para probar el motor de base de datos se realizó una pequeña prueba de llenado de formulario a través de la **WEB**, el formulario tiene 4 campos que se guardarán en la base de datos, esto nos dará una idea de cómo se conecta **PHP** con la base de datos y como se direccionan los datos a la misma.

La base de datos se crea a través del gestor de base de datos **PhpMyAdmin**. **PhpMyAdmin**, se ejecuta a través del navegador una vez iniciado los servicios de **wampserver**, el usuario y clave por defecto de **PhpMyAdmin** es **root** y **root**. La base de datos para este ejemplo será llamado **prueba** y tendrá una tabla llamada **persona**, donde esta contendrá cuatro columnas para almacenar los campos introducidos en el formulario. A continuación, se muestra la estructura de la base de datos.



Las cuatro columnas o campos es donde se almacenara los datos introducidos por el usuario, sin embargo, el primer campo **id_persona** no será introducida por el usuario, ya que, este campo se autoincrementara automáticamente dándole un numero único a cada persona registrada por el usuario.

Ahora bien, el código para llenar el formulario y guardarla en la base de datos esta separados en **tres (3) partes**:

1. El código principal, el cual, se encuentra programado el formulario que será llenado por el usuario.

El código principal se guarda con el nombre **índex**, esto es con la finalidad de que el navegador se dirija al código principal, en este mismo sentido, cada código **PHP** debe ser guardado con la extensión **.PHP**. El código principal, es básicamente un formulario realizado en **HTML (PHP** soporta **HTML** sin problema). A continuación se muestra el código principal.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
1. <?php
2.
3. ?>
4.
5. <!--Se Llama el archivo registros.php, el cual, tiene las
   consulta de mysql
6.           y la adquisición de datos se realiza a traves del metodo
   POST, este formulario esta realizado bajo HTML          -->
7. <form action="registros.php" method="post" name="form">
8. Nombres: <br/> <!--Se muestra en la pagina Nombres y el br es
   para salto de linea -->
9. <input type = "text" name="nombres"/> <br/> <!-- Entrada tipo
   texto y lo guarda en la variables nombres -->
10.    Apellido Paterno: <br/>
11.    <input type = "text" name="a_paterno"/> <br/>
12.    Apellido Materno: <br/>
13.    <input type = "text" name="a_materno"/> <br/>
14.    Fecha de Nacimiento: <br/>
15.    <input type = "date" name="fecha_nac"/> <br/> <!-- Entrada
   tipo date y lo guarda en la variables nombres -->
16.    <input type = "submit" value="Guardar Datos"/><!-- Entrada
   tipo enviar (boton para enviar los datos), envia los datos al
   archivo registros.php -->
17.    </form>
```

En la línea de código **7** se puede observar que se llama al archivo “**registros.php**” y se usara el método **POST** para pasarle los datos introducidos por el usuario a este archivo llamado “**registros.php**” (es el código que se encarga de realizar la consulta a **MySQL** y guardar los datos). Seguidamente, se observa campos de entradas tipo texto (línea **9, 11, 13, 15, 16**) y se guardan en las siguientes variables; **nombres, a_paterno, a_materno, fecha_nac** y por último el botón de guardar los datos, el cual, envía los datos al archivo “**registros.php**” mediante el método **POST**. Cabe destacar que, las variables deben tener el mismo nombre en ambos archivos para así evitar conflictos y/o confusiones.

Nota: El método post se usa cuando se quiere pasar datos de manera interna, de este modo el usuario no puede visualizar los datos que se envían al archivo **registros.php**, el método **GET** envía los datos a través del **URL** y el usuario puede observar los datos pasados al archivos, ambos métodos son efectivos, sin embargo, depende la necesidad del programador cual método utilizar.

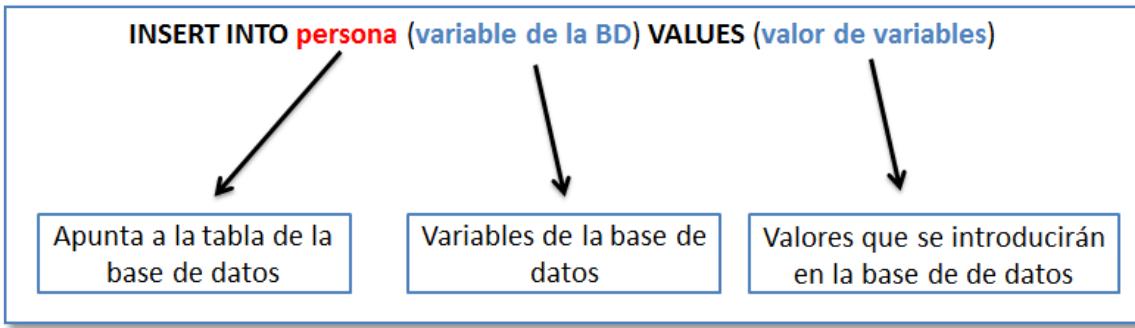
En la siguiente figura se observa la interfaz que se muestra al usuario para ingresar los datos, dicha interfaz se realizó con el código descrito anteriormente.

The screenshot shows a web browser window titled "localhost" with the URL "localhost/pru". The page displays a form with five fields: "Nombres" (Name), "Apellido Paterno" (Last Name), "Apellido Materno" (Last Name), "Fecha de Nacimiento" (Date of Birth), and a date input field with the placeholder "dd / mm / aaaa". Below the input fields is a "Guardar Datos" (Save Data) button.

2. El código intermedio, es el que realiza la consulta a la base de datos MySQL, es decir, este código toma los datos introducido por el usuario y los guarda en la base de datos. El código está guardado con el nombre “**registros.php**” tal como se ha mencionado en el apartado anterior. A continuación se muestra el código donde se realiza la consulta a la base de datos.

```
3.  
4. <?php  
5. include 'conexion.php'; // Llama al archivo conexion mediante el  
   include  
6.      ///Recibir los datos mediante POST y almacenarlos en  
   variables  
7. $nombres = $_POST["nombres"]; // Se definen las variables, cabe  
   destacar que, las variables tienen que coincidir con el nombre  
   de las variables definida en el formulario.  
8. $a_paterno = $_POST["a_paterno"];  
9. $a_materno = $_POST["a_materno"];  
10.    $fecha_nac = $_POST["fecha_nac"];  
11.           //Se guarda en la variable $insertar, la consulta  
   para agregar los valores a la base de datos.  
12.    $insertar= "INSERT INTO persona(nombres, a_paterno,  
   a_materno, fecha_nac) VALUES  
   ('$nombres', '$a_paterno', '$a_materno', '$fecha_nac')";  
13.           //Se guarda en la variable $resultado la Ejecución  
   de la consulta, llamando a la variable $conexion e $insertar.  
14.           //La variable $conexion esta en el archivo  
   conexion.php, de este modo se conecta a la base de datos y se  
   realiza la consulta.  
15.           //En la variable $conexion se encuentra el usuario  
   y password para entrar a la base de datos y en variable  
   $insertar se encuentra lo que se quiere agregar en la base de  
   datos.  
16.    $resultado= mysqli_query ($conexion,$insertar);  
17.    if(!$resultado)  
18.        {  
19.            echo 'Error al registrarse';  
20.        }  
21.    else  
22.        {  
23.            echo '<script>  
24.                alert("Persona Registrada Exitosamente");  
25.                window.history.go(-1);  
26.            </script>';  
27.        }  
28.    //Cerrar conexion  
29.    mysqli_close ($conexion);  
30.    ?>
```

Se puede observar que, el código realiza un llamado o un include del archivo **conexion.php** para conectar con la base de dato especificada dentro dicho archivo “**conexion.php**” (este código se explicara en el siguiente apartado). Esencialmente el código obtiene los datos introducido en el formulario (anteriormente explicado) mediante el método POST, para luego guardar los datos en las variables definidas (**\$nombres**, **\$a_paterno**, **\$a_materno**, **\$fecha_nac**) y ejecutar la consulta para agregar los valores a la base de datos. La estructura de la consulta es la siguiente:

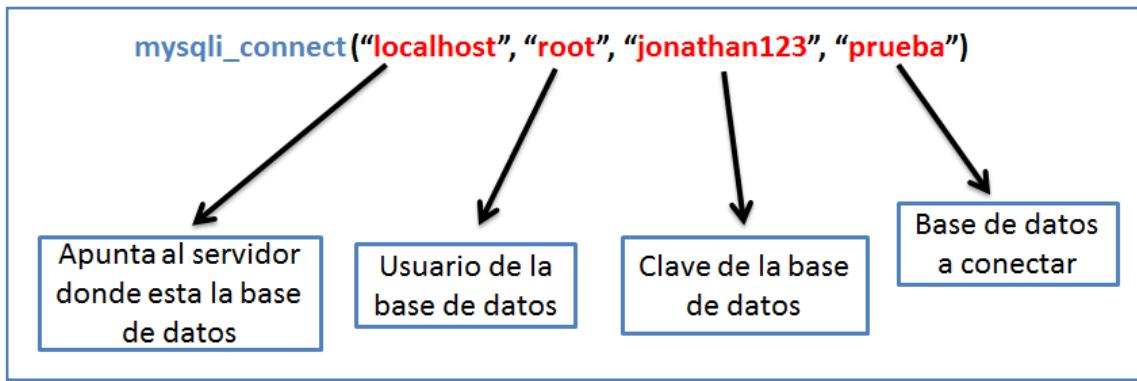


En la consulta se apunta a la tabla de la base de datos, de igual manera, se apunta a las columnas de las variables de la base de datos y finalmente se pasa el valor de cada variable.

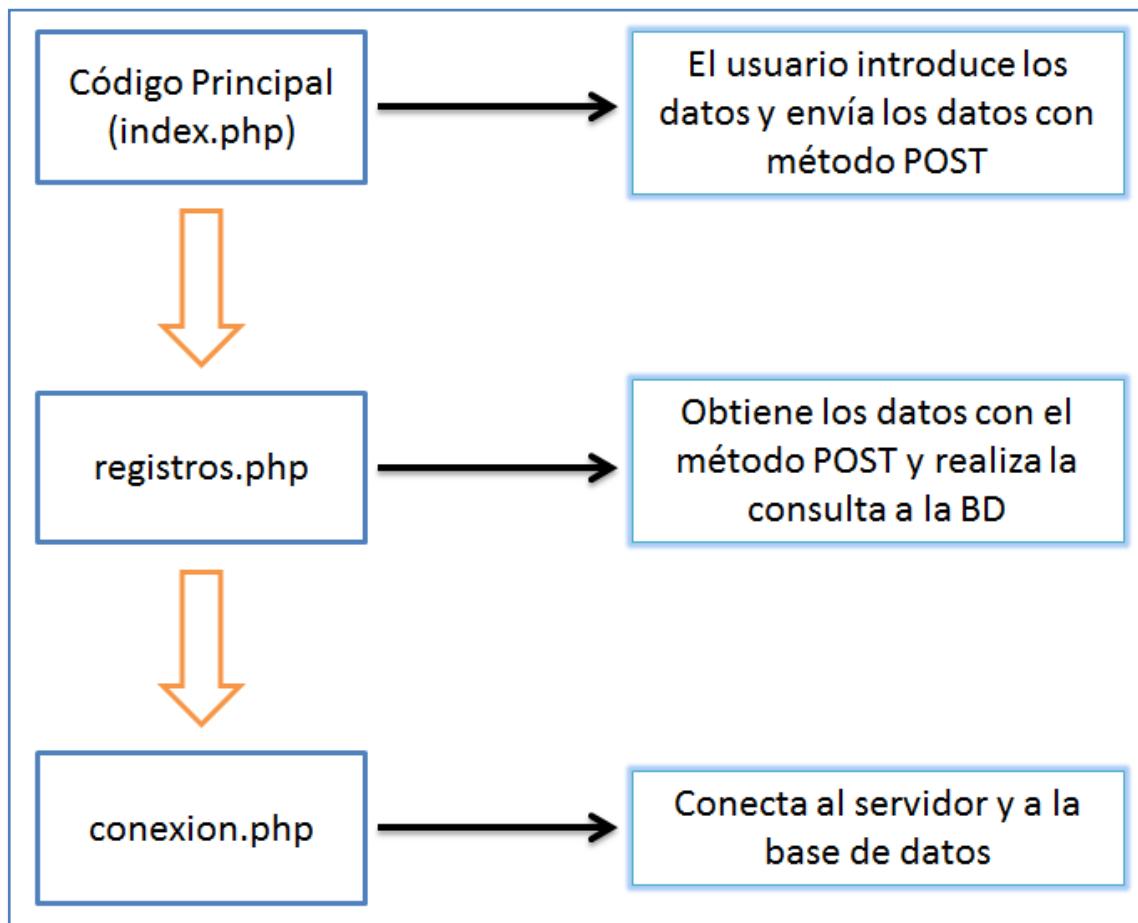
3. Por ultimo tenemos el código que realiza la conexión a la base de datos, dicho código es llamado cada vez que queremos realizar una consulta a la base de datos. En el código se define la base de datos a conectar, usuario, clave y la localización del servidor, el cual, para este es **localhost**. A continuación se muestra el código de conexión a la base de datos.

```
4. <?php
5. $conexion = mysqli_connect("localhost", "root", "mauricio123", "pr
ueba"); // conexión, usuario, contraseña, base de dato a
conectar
6.         /*if(!$conexion)
7.         {
8.             echo 'Error al conectar a la base de datos';
9.         }
10.        else
11.        {
12.            echo 'Conectado a la base de datos';
13.        }*/
14.    ?>
```

El código básicamente realiza la conexión a la base de datos mediante los parámetros pasados a la función (host o servidor, usuario, contraseña). A continuación se describe la estructura de la función.



En resumen, el código principal (`index.php`) recibe los datos a través del formulario, pasa los datos a través del método **POST** al archivo o código `registros.php` y por último el código `registros.php` llama al archivo `conexion.php` para conectar a la base de datos deseada.



METODO GET Y POST

Cuando un usuario rellena un formulario en una página web los datos hay que enviarlo de alguna manera, para enviarlo se pueden utilizar los métodos **GET** y **POST**.

El método **GET** utiliza la URL para enviar los datos del formulario. El formulario consta de varios campos que al usuario se le solicitan como nombre, apellidos, correo electrónico y mensaje.

Formulario Usuario

Nombre

Apellido

Email

País

Sexo

El resultado usando el método **GET** al enviar los datos seria el siguiente:

[http://www.Maverdug.com/newuser.php?nombre=pepe&apellido=flores&email=pepe%web.es&sexo=mujer.](http://www.Maverdug.com/newuser.php?nombre=pepe&apellido=flores&email=pepe%web.es&sexo=mujer)

En la **URL** se pueden distinguir varias partes:

httpd://www.Maverdug.com/newuser.php es la dirección web, el símbolo “?” indica desde donde empiezan los parámetros que se han enviado a través del formulario. Después del símbolo “?” se pueden apreciar parejas de datos con su nombre y valor separados por el símbolo “&”. Las parejas dato1=valor1, dato2=valor2 y dato3=valor3, reflejan el nombre y el valor de los campos enviados por el formulario.

Es decir, nombre=pepe, apellido=flores etc., esto nos indica que el campo del formulario que se denomina **nombre** llega con el valor “**pepe**” mientras que el campo del formulario que se denomina **apellidos** llega con el valor “**flores**”. Se tiene que tener en cuenta que para separar la primera pareja de la dirección web, se usa el símbolo “?” y para separar las restantes parejas entre sí, se usa el símbolo “&”.

En el caso de un envío de datos usando el método **POST**, estos no se envían a través de la **URL** si no de manera oculta, es decir, no es visible para el usuario los datos enviados. La diferencia entre ambos métodos radica en la forma de enviar los datos a la página cuando se pulsa el botón “Enviar” o “Guardar”. Tanto **GET** como **POST** son métodos de envíos válidos y ampliamente utilizados, elegir entre un método y otro depende de la aplicación concreta que se esté desarrollando.

COMO ACCEDER A WAMPSERVER DESDE OTRA RED

Para accede al servidor **Wampserver 3.0.6** desde otra red local o externa al servidor, se debe configurar el servidor a través de los archivos interno. Esto es un paso importante a realizar, ya que, el **arduino** necesita acceder al servidor desde una red externa, por ende, si no realizamos este paso el servidor bloqueara la petición del **arduino**. La ruta del archivo que se tiene que modificar es la siguiente:

C:\wamp64\bin\apache\apache2.4.23\conf\extra.

En esta dirección se encuentra el archivo **httpd-vhosts.conf** y modificamos las siguientes líneas para dar acceso al servidor de manera externa.

- **Requiere local:** Solo permite acceso desde el host.
- **Requiere all granted:** Permite acceso desde otra Red.

Solo comentamos la línea que permite el acceso desde el host y añadimos la línea **Require all granted** como se muestra a continuación.

```
1 # Virtual Hosts
2 #
3
4 <VirtualHost *:80>
5   ServerName localhost
6   DocumentRoot c:/wamp64/www
7   <Directory "c:/wamp64/www/">
8     Options +Indexes +Includes +FollowSymLinks +MultiViews
9     AllowOverride All
10    Require all granted → Permite conexiones externas
11    #Require local → Permite acceso solo desde el Host
12    #Deny from IP
13  </Directory>
14 </VirtualHost>
15 #
16
17 #Si Se requiere denegar el permiso de entrar al servidor wampserver.
18 #Desde otra red, descomentamos la Linea 11 (Require local) y comentamos la linea 10 (Require all granted).
19 #Require local nos dara acceso solo a red del host y Require all granted nos dara acceso desde red externa.
20
21 #Para denegar acceso de una IP especifica en el comando Deny from IP, colocamos la ip que necesitamos bloquear en IP
```

Nota: Es importante destacar que, esta modificación se realiza cuando se tienen **host virtual**, en caso de no tener host virtual se tienen que dirigir al archivo **httpd.conf** ubicado en la siguiente ruta **C:\wamp64\bin\apache\apache2.4.23\conf** y ahí identificar las líneas que permite modificar el acceso al servidor (Require local, Requiere all granted).

Nota: Para acceder a **PhpMyadmin** desde una red externa se realiza los mismo pasos, pero, en el archivo ubicado en la siguiente ruta, **C:\wamp64\alias** y abrimos el archivo **PphpMyadmin.conf** para modificar las misma líneas mencionada en el apartado anterior (Require local, Requiere all granted). Esto se realiza si se quiere visualizar la base de datos desde otra red, sin embargo, no interfiere con las peticiones que realiza el arduino al servidor.

RUTINA ENVIAR DATOS A MYSQL

A continuación se muestra la rutina realizada en el arduino para enviar datos a **Mysql**, así como también, se mostrara el código realizado en **PHP** para guardar los datos enviados por el arduino a la base de datos de **Mysql**.

Primeramente se muestra el código realizado en **PHP** para tomar el dato enviado por el arduino y guardarlo en la base de datos.

```
1. <?php
2. include 'conexion.php';
3. //Recibir los datos y almacenarlos en variables
4. $temperatura = $_GET["temperatura"];
5. // Consulta para insertar
6. $insertar= "INSERT INTO sensores(temperatura) VALUES
('$temperatura')";
7. // Ejecutar consulta
8. mysqli_query ($conexion,$insertar); // la variable conexion esta
en el archivo conexion.php, de este modo se conecta a la base de
datos y se realiza la consulta. En la variable conexión se
encuentra el usuario y password para entrar a la base de datos y
en la otra variable se encuentra lo que se quiere insertar en la
base de datos
9.
10.      ?>
```

El código realiza la toma de datos mediante el método **GET** y seguidamente realiza la consulta a la base de datos para guardar el valor enviado. La dirección web o el URL para enviar el dato al servidor es la siguiente:

- **190.215.92.114/arduino/registros.php?temperatura=18**

Es decir, si se coloca esa dirección en la URL del navegador se insertara el valor 18 a la base de dato estructurada para el almacenaje de la temperatura. Como se puede observar en el código **PHP** la tabla donde se insertara los datos se llama sensores, sin embargo, la

base de datos descrita para ello es llamada arduino, este no se aprecia, ya que, está dentro el archivo conexión, el cual, es llamado desde el código principal mostrado anteriormente. A continuación se muestra el código en **PHP** del archivo **conexión.php**.

```

1. <?php
2. $conexion = mysqli_connect("localhost", "root", "mauricio123", "ar
   duino"); // conexión, usuario, contraseña, base de dato
3.         /*if(!$conexion)
4.         {
5.             echo 'Error al conectar a la base de datos';
6.         }
7.     else
8.     {
9.         echo 'Conectado a la base de datos';
10.    }*/
11.    ?>

```

Se puede observar que, la base de datos donde conecta es llamada arduino. A continuación, se observa el inserto del valor hecho a través del navegador.

The screenshot shows the MySQL Workbench interface. On the left, the 'Base de Datos' (Database) pane lists databases like 'arduino', 'sensores', and 'information_schema'. A red arrow points from the 'arduino' database to a blue box labeled 'Base de Datos'. Another red arrow points from the 'sensores' table to a blue box labeled 'Tabla'. On the right, the 'Valor enviado' (Value sent) pane displays a query result for the 'sensores' table. It shows one row with values: 'temperatura' (18) and 'FECHA Y HORA' (2017-01-20 16:43:35). A red box highlights this row, and a red arrow points from it to a blue box labeled 'Columnas o variables de la tabla' (Columns or variables of the table).

La estructura de la base de datos está compuesta simplemente por dos columnas o variables a guardar, la principal es la “**temperatura**” y la segunda columna es la **fecha y**

hora, esta última variable no es enviada, ya que, el servidor toma la fecha y hora automáticamente en la que se insertó el valor “**temperatura**”.

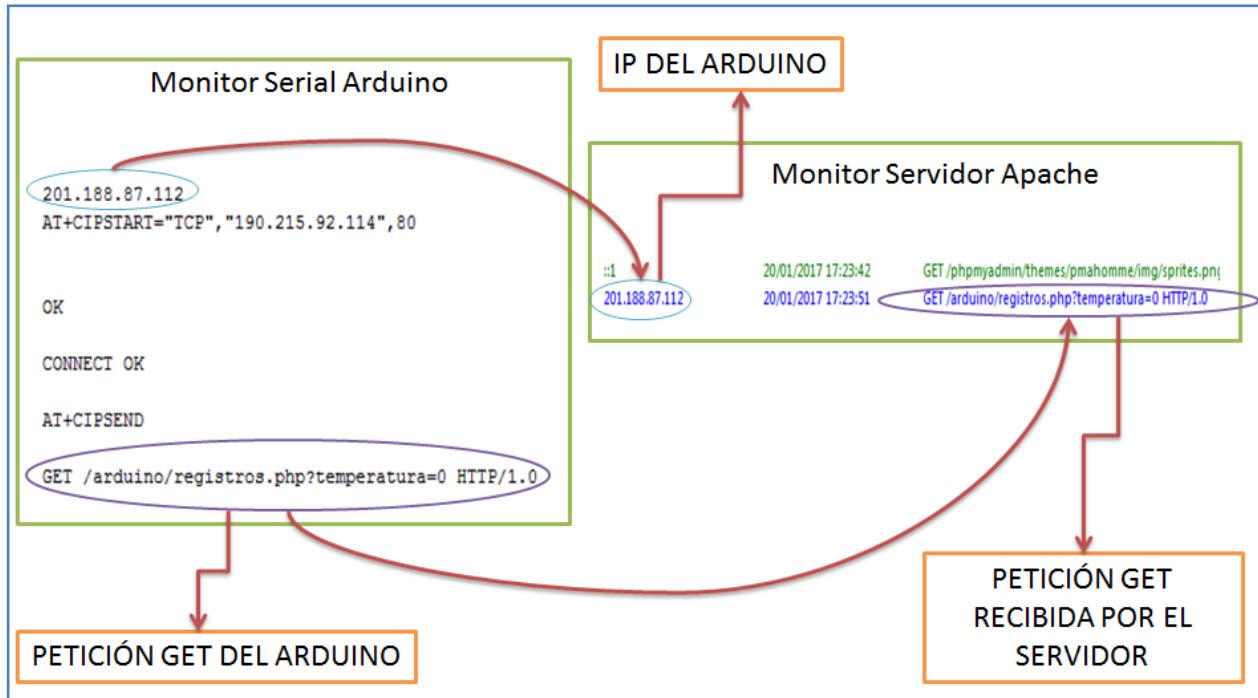
Ahora bien, la misma petición que se realizó mediante el navegador se hará con el arduino, pero, añadiéndole la palabra **GET** y de igual manera el protocolo o versión de **HTTP** utilizado para enviar la petición. A continuación, se muestra el código para enviar la petición **GET** desde el arduino e insertar un valor contenido en la variable **X**, el cual, dicho valor será **0**.

```
1. ///////////////
2. int i=0,mostrar=0,e=0,x=0;
3. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
4. void setup()
5. {
6.   i=0;
7.   Serial.begin(19200);
8.   delay(25000); //tiempo para ingresar a la red telefónica
9. }
10. void loop()
11. {
12.   if(i==0)
13.   {
14.     conectgprs(); // Función conectar GPRS se ejecuta solo
   cuando i=0
15.   }
16.
17. }
18. ////////////////////FUNCIÓN CONECTAR A
   GPRS///////////////////////////////
19. void conectgprs()
20. {
21.   if (Serial.available()==0) //Esperar Serial disponible
22.   {
23.     Serial.println("AT+CGATT?\r\n"); // Comando AT para
   entrar a conexión TCP
24.     clearBuffer();
25.     delay(3000);
26.     Serial.println("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\""
   \r\n"); // Configurar APN, USUARIO y CONTRASEÑA
27.     clearBuffer();
28.     delay(3000);
29.     Serial.println("AT+CIICR\r\n"); // Levantar conexión GPRS
30.     clearBuffer();
31.     delay(3000);
32.     Serial.println("AT+CIFSR\r\n"); // Obtener dirección IP
33.     clearBuffer();
34.     delay(3000);
35.     Serial.println("AT+CIPSTART=\"TCP\",\"190.215.92.114\",80
   \r\n"); // Conectarse al servidor mediante TCP
36.     clearBuffer();
37.     delay(3000);
```

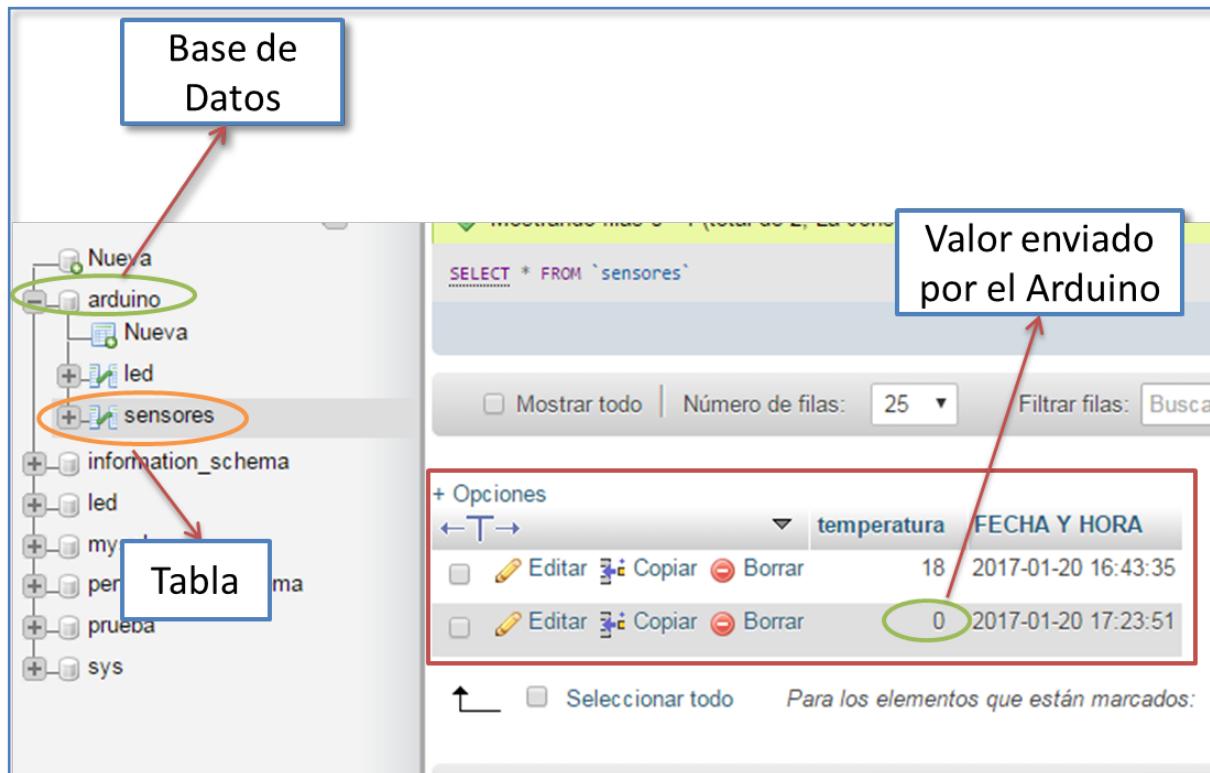
BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
38.         Serial.println("AT+CIPSEND\r\n"); // Preparar el mensaje
    a enviar
39.         delay(3000);
40.         Serial.print("GET
    /arduino/registros.php?temperatura=");
    // Escribir mensaje a
    enviar al servidor mediante el metodo GET, se envia sin fin de
    linea para poder enviar el dato "x", luego de enviar el dato se
    envia fin de linea y retorno de carro "\r\n"
41.         delay(100);
42.         Serial.print(x);
43.         delay(100);
44.         Serial.print(" HTTP/1.0"); // Se envia la version del
    protocolo, para este caso HTTP/1.0
45.         delay(3000);
46.         Serial.println("\r\n"); // Entre linea
47.         delay(3000);
48.         Serial.println("\xA1"); // Caracter especial para enviar
    mensaje al servidor
49.         delay(3000);
50.         i=1;
51.     }
52.
53. }
54. ///////////////////////////////////////////////////
55.
56. ////////////////LIMPIAR
    BUFFER///////////////
57. void clearBuffer()
58. {
59.
60.     while (Serial.available() !=0)
61.     {
62.         Serial.println(Serial.readString()); // Leer respuesta
    del sim900
63.         Serial.flush(); // Liberar buffer
64.         Clear=Serial.readStringUntil('\n'); // Limpiando Buffer
65.         Clear=""; // Iniciar variable en 0
66.     }
67. }
68. ///////////////////////////////////////////////////
    /
```

El código es básicamente igual al que se utilizó anteriormente para conectarse al servidor simulado **HERCULES**, sin embargo, en este enviamos una petición **GET** al servidor **Apache** por el puerto **80** y usando el protocolo de conexión **HTTP 1.0**. Es importante escribir el protocolo utilizado luego de hacer la petición **GET**, ya que, si no lo escribimos el servidor rechazará la petición **GET**. La petición **GET** realizada es la siguiente, “**GET /arduino/registros.php?temperatura=x**” y seguidamente se añade el la versión del protocolo en este caso **HTTP 1.0**.



En la imagen se puede observar, como el **Arduino** obtiene la dirección **IP** y luego de estar conectado al servidor de manera exitosa envía la petición **GET** con el valor **0**, el cual, es el valor que se va almacenar en la base de datos. A continuación se muestra el valor recibido en la base de datos.



RUTINA ENVIAR DATOS A MYSQL Y VERIFICAR CONEXIÓN

A continuación, se muestra la rutina para enviar datos a **MYSQL** y verificar la conexión con el servidor.

```
1. ///////////////
2. int i=0,mostrar=0,e=0,x=0;
3. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
4. void setup()
5. {
6.   i=0;
7.   Serial.begin(19200);
8.   delay(4000); //tiempo para ingresar a la red telefónica
9. }
10.    void loop()
11.    {
12.      if(i==0)
13.      {
14.        conectgprs(); // Función conectar GPRS se ejecuta solo
   cuando i=0
15.      }
16.      if (i==1)
17.      {
18.        Read(); //Función Read se ejecuta cuando i=1
19.      }
20.
21.    }
22.    ////////////////////FUNCIÓN CONECTAR A
   GPRS///////////////////////////////
23.    void conectgprs()
24.    {
25.      if (Serial.available()==0)// Serial Disponible
26.      {
27.        Serial.println("AT+CGATT?\r\n"); // Comando AT para
   entrar a conexión TCP
28.        clearBuffer();
29.        delay(3000);
30.        Serial.println("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\"
   \r\n"); // Configurar APN, USUARIO y CONTRASEÑA
31.        clearBuffer();
32.        delay(3000);
33.        Serial.println("AT+CIICR\r\n"); // Levantar conexión GPRS
34.        clearBuffer();
35.        delay(3000);
36.        Serial.println("AT+CFUSR\r\n"); // Obtener dirección IP
37.        clearBuffer();
38.        delay(3000);
39.        Serial.println("AT+CIPSTART=\"TCP\",\"190.215.92.114\",80
   \r\n"); // Conectarse al servidor mediante TCP
40.        clearBuffer();
41.        delay(3000);
42.        Serial.println("AT+CIPSEND\r\n"); // Preparar el mensaje
   a enviar
43.        delay(3000);
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
44.      Serial.print("GET
/arduino/registros.php?temperatura="); // Escribir mensaje a
enviar
45.      delay(100);
46.      Serial.print(x);
47.      delay(100);
48.      Serial.print(" HTTP/1.0");
49.      delay(3000);
50.      Serial.println("\r\n"); // Entre linea
51.      delay(3000);
52.      Serial.println("\x1A"); // Caracter especial para enviar
mensaje al servidor
53.      clearBuffer();
54.      clearBuffer();
55.      i=1;
56.    }
57.
58.  }
59. ///////////////////////////////////////////////////////////////////
60.
61. //////////////////////////////////////////////////////////////////LIMPIAR
BUFFER/////////////////////////////////////////////////////////////////
62. void clearBuffer()
63. {
64.
65.   while (Serial.available() !=0)
66.   {
67.     // Serial.println(Serial.readString()); Usado para cuando
se quiere leer a respuesta del sim900, para ello, se tiene que
comentar las 3 lineas siguientes y descomentar esta linea.
68.     Serial.flush();
69.     Clear=Serial.readStringUntil('\n'); // Limpiando Buffer,
Guardando respuesta en la variable Clear
70.     Clear=""; // Iniciar variable en 0
71.   }
72. }
73. ///////////////////////////////////////////////////////////////////
74. //////////////////////////////////////////////////////////////////VERIFICACION DE
CONEXION/////////////////////////////////////////////////////////////////
75. void Read()
76. {
77.   if(Serial.available()==0) // Espera el Serial Disponible
78.   {
79.     Serial.println("AT+CIPSTATUS\r\n"); // Envia el
comando Status para verificar conexión
80.     delay(100);
81.     delay(100);
82.     while (Serial.available() !=0)// Mientras el buffer
este lleno de datos se queda en while
83.     {
84.       control=Serial.readStringUntil('\n');// Lee la
respuesta del modulo SIM900
85.       delay(25);
86.       control1=control; // Pasamos la variable control a
control1 para luego compararla
87.       control.trim(); // Corta los espacio del string
88.       control1.trim();
89.       Serial.println(control1);
```

```
90.          }
91.          if(control1.equals(control2)) // Compara la
   respuesta con control2 = "STATE: CONNECT OK"
92.          {
93.              Serial.write("Conexion Exitosa");
94.              i=2;
95.          } else i=0;
96.
97.      }
98.      clearBuffer(); // Limpiar Buffer
99.      control=""; // Iniciar la variable en 0 para la proxima
   comparación
100.     control1="";
101. }
102. //////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
```

El código básicamente es igual, al código “[RUTINA DE CONEXIÓN TCPIP CON VERIFICACIÓN DE CONEXIÓN AL SERVIDOR](#)”, por ende, no se explicara a detalle.

RUTINA ENVIAR DATOS MYSQL, VERIFICACIÓN DE CONEXIÓN CADA DOS MINUTOS

A continuación, se muestra la rutina para enviar datos a **MYSQL** y verificar la conexión cada 2 minutos.

```
1. //////
2. int i=0,mostrar=0,e=0,x=0,segundos=0,ia=0;;
3. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
4. void setup()
5. {
6.     i=0;
7.     Serial.begin(19200);
8.     delay(10000); //tiempo para ingresar a la red telefónica
9. //////////////////////////////////////////////////////////////////INTERRUPCIONES////////////////////////////////////////////////////////////////
   /**
10.     cli(); // Deshabilitar interrupciones globales
11.     TCCR4A=0; //Iniciando Registro
12.     TCCR4B=0; //Iniciando Registro
13.     OCR4A=15624; // Valor el cual timer se va a comparar con
   el CTC para reiniciar en 1 Seg
14.     TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
15.     TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
   con el CS42
16.     TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
   con el CS40
17.     sei(); // Activar las interrupciones
18. //////////////////////////////////////////////////////////////////
   /**
19. }
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
20.      void loop()
21.    {
22.      if(i==0)
23.      {
24.        conectgprs(); // Función conectar GPRS se ejecuta solo
cuando i=0
25.      }
26.      if (i==1)
27.      {
28.        Read(); //Función Read se ejecuta cuando i=1
29.      }
30.      if(i==2)
31.      {
32.        segundos=0;
33.        TIMSK4 =(1<<OCIE4A); // activando interrupción por
comparación
34.        i=3;
35.      }
36.      if (i==3)
37.      {
38.        enviardatoAnalog(ia); // Llama a la función enviar
dato analógico, AIO
39.      }
40.    }
41.    ////////////////////FUNCIÓN CONECTAR A
GPRS/////////////////////////////
42.    void conectgprs()
43.    {
44.      TIMSK4 =(0<<OCIE4A);
45.      if (Serial.available()==0)
46.      {
47.        Serial.flush();
48.        Serial.println("AT+CGATT?\r\n"); // Comando AT para
entrar a conexión TCP
49.        clearBuffer();
50.        delay(1000);
51.        Serial.println("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\"
\r\n"); // Configurar APN, USUARIO y CONTRASEÑA
52.        clearBuffer();
53.        delay(1000);
54.        Serial.println("AT+CIICR\r\n"); // Levantar conexión GPRS
55.        clearBuffer();
56.        delay(1000);
57.        Serial.println("AT+CIFSR\r\n"); // Obtener dirección IP
58.        clearBuffer();
59.        delay(1000);
60.        Serial.println("AT+CIPSTART=\"TCP\",\"190.215.92.114\",80
\r\n"); // Conectarse al servidor mediante TCP
61.        clearBuffer();
62.        delay(300);
63.        clearBuffer();
64.        i=1;
65.      }
66.
67.    }
68.    /////////////////////////////////
69.    //////////////////LIMPIAR
70.    ////////////////////BUFFER/////////////////////////////
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
71.      void clearBuffer()
72.    {
73.
74.      while (Serial.available() !=0)
75.      {
76.        // Serial.println(Serial.readString()); Usado para cuando
77.        // se quiere leer a respuesta del sim900, para ello, se tiene que
78.        // comentar las 3 lineas siguientes y descomentar esta linea.
79.        Serial.flush();
80.        Clear=Serial.readStringUntil('\n'); // Limpiando Buffer,
81.        // Guardando respuesta en la variable Clear
82.        Clear=""; // Iniciar variable en 0
83.      }
84.      /////////////////////////////////
85.      void Read()
86.    {
87.      TIMSK4 =(0<<OCIE4A);
88.      if(Serial.available()==0) // Espera el Serial Disponible
89.      {
90.        Serial.println("AT+CIPSTATUS\r\n"); // Envia el
91.        comando Status para verificar conexión
92.        delay(100);
93.        while (Serial.available() !=0) // Mientras el buffer
94.          este lleno de datos se queda en while
95.        {
96.          control=Serial.readStringUntil('\n');// Lee la
97.          respuesta del modulo SIM900
98.          delay(25);
99.          control1=control; // Pasamos la variable control a
100.         control1 para luego compararla
101.         control.trim(); // Corta los espacio del string
102.         control1.trim();
103.         Serial.println(control1);
104.         }
105.         if(control1.equals(control2))// Compara la
106.           respuesta con control2 = "STATE: CONNECT OK"
107.           {
108.             i=2;
109.           }
110.           }
111.           control=""; // Iniciar la variable en 0 para la proxima
112.           comparación
113.           control1="";
114.           ///////////////////////////////
115.           //////////////Función Interrumpir para verificar
116.           // Status///////////////////
117.           ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
118.           función cuando la comparación del CTC y el timer es 15624
```

```
117.    {
118.        segundos++;
119.        if(segundos==120) // Entra a la condición luego de 2
minutos.
120.    {
121.        i=1;
122.        segundos=0;
123.    }
124.    }
125.    /////////////////////////////////
126.    //////////////////////////////Enviar Datos
Analogicos/////////////////////////
127.    void enviardatoAnalog(int y)
128.    {
129.        if(Serial.available()==0)
130.        {
131.            x=analogRead(y); //Se lee el puerto analógico a traves
por paso de parametro de la función en este caso el puerto
analógico 0
132.            Serial.println("AT+CIPSEND\r\n"); // Preparar el mensaje
a enviar
133.            delay(1000);
134.            Serial.print("GET
/arduino/registros.php?temperatura="); // Escribir mensaje a
enviar al servidor mediante el metodo GET, se envia sin fin de
linea para poder enviar el dato "x", luego de enviar el dato se
envia fin de linea y retorno de carro "r\n"
135.            delay(300);
136.            Serial.print(x);
137.            delay(300);
138.            Serial.print(" HTTP/1.1\r\n"); // Se envia la version del
protocolo, para este caso HTTP/1.1
139.            delay(1000);
140.            Serial.print("Host: 190.215.92.114\r\n"); // De acuerdo
al protocolo HTTP/1.1 en la cabecera debe especificar el host
receptor
141.            delay(1000);
142.            Serial.print("Connection: Keep-Alive"); // El protocolo
HTTP/1.1 permite conexiones persistente, es decir, la conexion
queda abierta durante el tiempo programado en el servidor. Pues,
ya que el servidor responde a la peticion y cierra la
conexion.Sin embargo, esta configuracion se realizo en el
servidor y se puede omitir esta linea de codigo
143.            delay(1000);
144.            Serial.println("\r\n"); // Entre linea
145.            delay(1000);
146.            Serial.println("\xA"); // Caracter especial para enviar
mensaje al servidor
147.            clearBuffer();
148.        }
149.    }
```

El código básicamente es la esencia de “[Rutina Control De Conexión Interrupción](#)”, por ende, no se explicara a detalle.

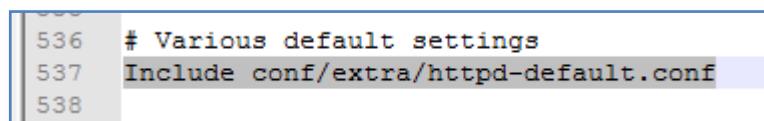
Nota: El protocolo **HTTP 1.0** no soporta conexión persistente, es decir, cuando se termina de realizar la petición **GET** o enviar el dato, el servidor cierra dicha conexión automáticamente, pues, si se requiere enviar nuevamente una solicitud **GET** se debe reconectar nuevamente al servidor. Ahora bien, para dejar la conexión abierta se usa el protocolo **HTTP 1.1**, este protocolo soporta conexiones persistentes y Multi-Host. Para enviar la solicitud **GET** con el protocolo **HTTP 1.1** se tiene que especificar el **HOST** en la cabecera de la petición **GET**, de la siguiente manera:

```
1. Serial.print("GET /arduino/registros.php?temperatura=");
2.   delay(300);
3.   Serial.print(x);
4.   delay(300);
5.   Serial.print(" HTTP/1.1\r\n"); // Se envia la version del
   protocolo, para este caso HTTP/1.1
6.   delay(1000);
7.   Serial.print("Host: 190.215.92.114\r\n"); // De acuerdo al
   protocolo HTTP/1.1 en la cabecera debe especificar el host
   receptor
8.   delay(1000);
9.   Serial.print("Connection: Keep-Alive"); // El protocolo
   HTTP/1.1 permite conexiones persistentes, es decir, la conexión
   queda abierta durante el tiempo programado en el servidor. Pues,
   ya que el servidor responde a la petición y cierra la
   conexión. Sin embargo, esta configuración se realizó en el
   servidor y se puede omitir esta línea de código
```

Con la línea **9** del código (“**Keep-Alive**”) se especifica que se dejara la conexión abierta, sin embargo, esto se puede realizar a través del servidor y omitir esta línea de código. Otro punto importante a destacar es que, en el servidor se debe especificar cuantos datos se enviarán o recibirán dejando la conexión abierta (**MaxKeepAliveRequests**) y también se debe activar la conexión persistente en el servidor **Apache**.

Para activar la conexión persistente en el servidor **Apache versión 2.4.23**.

1. Se modifica el archivo **httpd.conf**, el cual, se encuentra en la siguiente ruta; **C:\wamp64\bin\apache\apache2.4.23\conf**. En este archivo se quita como comentario la línea 537 (**Include conf/extra/httpd-default.conf**), así en la configuración del servidor incluirá el archivo de configuración por defecto.



```
536 # Various default settings
537 Include conf/extra/httpd-default.conf
538
```

2. Seguidamente, se modifica el archivo **httpd-default.conf**, este archivo se encuentra en la siguiente ruta:

C:\wamp64\bin\apache\apache2.4.23\conf\extra.

En este archivo se habilita el **Keep-Alive**, colocándolo a **On**, se modifica el **KeepAliveTimeOut** a 20 segundos, la cual, es el tiempo que quedara abierta la conexión y por último se modifica **MaxKeepAliveRequest** a 100, es decir, recibirá 100 peticiones y luego cerrara la conexión. Es importante destacar que, el # indica que la línea esta como comentario.

```
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive On

#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100
#MaxKeepAliveRequests 10

#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 20
#KeepAliveTimeout 5
```

Nota: Si se requiere cerrar la conexión persistente se puede realizar a través del arduino sustituyendo “Connection: Keep-Alive” por “Connection: Keep-Alive,Close”. Esto cerrara la conexión.

**RUTINA ENVIAR DATOS MYSQL, CONTROL DE CONEXIÓN, APAGAR Y
PRENDER SIM900 EN CASO DE FALLA DE CONEXIÓN**

En esta rutina se envía datos a **Mysql**, al mismo tiempo, se verifica la conexión cada dos minutos mediante interrupción, y en caso de que la conexión al servidor falle cinco veces, apaga y prende el **SIM900** para iniciar nuevamente el proceso de conexión. En este mismo sentido, también se verifica que el envío de dato se haya realizado correctamente. A continuación se muestra el código.

```
1. // En esta version se usa el puerto serial 1 para comunicarse
   con el sim900
2. ///////////
3. int i=0,mostrar=0,e=0,x=0,segundos=0,ia=0,conexion=0,ErrorReply=
   0;
4. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
5. //String enviar="ERROR", controlEnviar="",controlEnviar1="";
6. void setup()
7. {
8.   i=0;
9.   Serial.begin(19200);
10.  Serial1.begin(19200);
11.  // onGPRS();
12.  delay(10000); //tiempo para ingresar a la red telefonica
13.  Serial.println("Serial ON\r\n");
14.  delay (100);
15.  ////////////////////CONFIGURANDO
   INTERRUPCIONES///////////////////////////////
16.  cli(); // Deshabilitar interrupciones globales
17.  TCCR4A=0; //Iniciando Registro
18.  TCCR4B=0; //Iniciando Registro
19.  OCR4A=15624; // Valor el cual timer se va a comparar con
   el CTC para reiniciar en 1 Seg
20.  TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
21.  TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
   con el CS42
22.  TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
   con el CS40
23.  sei(); // Activar las interrupciones
24.  /////////////////////////////////
   //////
25. }
26. void loop()
27. {
28.   Serial.flush();
29.   Serial1.flush();
30.   if(i==0)
31.   {
32.     conectgprs(); // Funcion conectar GPRS se ejecuta solo
   cuando i=0
33.   }
34.   if (i==1)
35.   {
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
36.         Read(); //Funcion Read se ejecuta cuando i=1
37.     }
38.     if(i==2)
39.     {
40.         segundos=0;
41.         TIMSK4 = (1<<OCIE4A); // activando interrupcion por
comparacion
42.         i=3;
43.     }
44.     if (i==3)
45.     {
46.         enviardatoAnalog(ia); // Llama a la funcion enviar
dato analogico, AIO
47.     }
48.     if(i==20)
49.     {
50.         offGPRS();
51.         delay(3000);
52.         onGPRS();
53.     }
54.
55. }
56. ////////////FUNCION CONECTAR A
GPRS///////////
57. void conectgprs()
58. {
59.     if(Serial.available()==0)// Verificar Serial disponible
60.     {
61.         TIMSK4 = (0<<OCIE4A); // Desactivar interrupcion
62.         Serial.println("Conectando..");
63.         delay(300);
64.         Serial1.print("AT+CGATT?\r\n"); // Comando AT para entrar
a conexion TCP
65.         clearBuffer();
66.         delay(1000);
67.         Serial1.print("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\"\r\n"); // Configurar APN, USUARIO y CONTRASEÑA
68.         clearBuffer();
69.         delay(1000);
70.         Serial1.print("AT+CIICR\r\n"); // Levantar conexion GPRS
71.         clearBuffer();
72.         delay(1000);
73.         Serial1.print("AT+CIFSR\r\n"); // Obtener direccion IP
74.         clearBuffer();
75.         delay(1000);
76.         Serial1.print("AT+CIPSTART=\"TCP\",\"190.215.92.114\",80\r\n"); // Conectarse al servidor mediante TCP
77.         clearBuffer();
78.         delay(300);
79.         clearBuffer();
80.         i=1;
81.     }
82. }
83. /////////////
84. ////////////LIMPIAR
BUFFER///////////
85. void clearBuffer()
86. {
87. }
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
88.
89.     while (Serial1.available() !=0)
90.     {
91.         Serial.println(Serial1.readString()); //Usado para cuando
92.         se quiere leer a respuesta del sim900, para ello, se tiene que
93.         comentar las 3 lineas siguientes y descomentar esta linea.
94.         Clear=Serial1.readStringUntil('\n'); // Limpiando Buffer,
95.         Guardando respuesta en la variable Clear
96.         Clear=""; // Iniciar variable en 0
97.     }
98.     /////////////////////////////////
99.     ///////////////VERIFICACION DE
100.    CONEXION///////////////////////
101.   void Read()
102.   {
103.       Serial.println("Verificando Conexion");
104.       if(Serial1.available()==0)
105.       {
106.           TIMSK4 = (0<<OCIE4A); // Desactivar interrupcion
107.           Serial1.println("AT+CIPSTATUS\r\n"); // Envia el comando
108.           Status para verificar conexion
109.           delay(100);
110.           while (Serial1.available()!=0)// Mientras el buffer
111.               este lleno de datos se queda en while
112.               {
113.                   control=Serial1.readStringUntil('\n');// Lee la
114.                   respuesta del modulo SIM900
115.                   delay(25);
116.                   control1=control; // Pasamos la variable control a
117.                   control1 para luego compararla
118.                   control.trim(); // Corta los espacio del string
119.                   control1.trim();
120.                   Serial.println(control);
121.                   }
122.                   if(control1.equals(control)) // Compara la
123.                   respuesta con control2 = "STATE: CONNECT OK"
124.                   {
125.                       i=2;
126.                       conexion=0;
127.                   } else
128.                   {
129.                       i=0;
130.                      conexion++;
131.                   }
132.                   if (conexion==5)
133.                   {
134.                       i=20;
135.                   }
136.               ///////////////////////////////
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
136.     //////////////Funcion Interrumpir para verificar
137.     Status///////////////////////////
138.     {
139.         segundos++;
140.         if(segundos==120)// Entr a la condicin luego de 2
141.             {
142.                 i=1;
143.                 segundos=0;
144.             }
145.         }
146.         /////////////////////////////////
147.         //////////////////////////////Enviar Datos
148.         Analogicos///////////////////////
149.         void enviardatoAnalog(int y)
150.         {
151.             if(Serial1.available()==0)
152.             {
153.                 x=analogRead(y); //Se lee el puerto analogico a traves
154.                 por paso de parametro de la funcion en este caso el puerto
155.                 analogico 0
156.                 Serial1.println("AT+CIPSEND\r\n"); // Preparar el mensaje
157.                 a enviar
158.                 delay(1000);
159.                 Serial1.print("GET
160.                 /arduino/registros.php?temperatura="); // Escribir mensaje a
161.                 enviar al servidor mediante el metodo GET, se envia sin fin de
162.                 linea para poder enviar el dato "x", luego de enviar el dato se
163.                 envia fin de linea y retorno de carro "\r\n"
164.                 delay(300);
165.                 Serial1.print(x);
166.                 delay(300);
167.                 Serial1.print(" HTTP/1.1\r\n"); // Se envia la version
168.                 del protocolo, para este caso HTTP/1.1
169.                 delay(1000);
170.                 Serial1.print("Host: 190.215.92.114\r\n"); // De acuerdo
171.                 al protocolo HTTP/1.1 en la cabecera debe especificar el host
172.                 receptor
173.                 delay(1000);
174.                 Serial1.print("Connection: Keep-Alive"); // El protocolo
175.                 HTTP/1.1 permite conexiones persistente, es decir, la conexion
176.                 queda abierta durante el tiempo programado en el servidor. Pues,
177.                 ya que el servidor responde a la peticin y cierra la
178.                 conexion.Sin embargo, esta configuracion se realizo en el
179.                 servidor y se puede omitir esta linea de codigo
180.                 delay(1000);
181.                 Serial1.println("\r\n"); // Entre linea
182.                 delay(1000);
183.                 Serial1.println("\xA1"); // Caracter especial para enviar
184.                 mensaje al servidor
185.                 WaitReply("ERROR","Envio Fallido"); // Paso el parametro
186.                 ERROR y el parametro Envio Fallido
187.             }
188.         }
189.         /////////////////////////////////
190.         ///////////////////////////////
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
172.      ///////////////// APAGAR GPRS EN CASO DE FALLAR
173.      CONEXION///////////
174.      void offGPRS ()
175.      {
176.          Serial.println("Reiniciando SIM900..."); 
177.          delay(3000);
178.          clearBuffer();
179.          if(Serial.available()==0)
180.          {
181.              Serial.println("AT+CPOWD=1");
182.              delay(100);
183.              clearBuffer();
184.          }
185.      }
186.      void onGPRS ()
187.      {
188.          Serial.println("Encendiendo SIM900..."); 
189.          delay(3000);
190.          pinMode (9,OUTPUT);
191.          digitalWrite (9, LOW);
192.          delay(1000);
193.          digitalWrite (9, HIGH);
194.          delay(2000);
195.          digitalWrite(9,LOW);
196.          delay(3000);
197.          i=0;
198.          conexion=0;
199.          delay(3000);
200.          clearBuffer();
201.      }
202.      /////////////////
203.      ////////////////FUNCION LEER RESPUESTA/////////////
204.
205.      void WaitReply(String x, String y)
206.      {
207.          String controlReply;
208.          String controlReply1;
209.          while (Serial1.available() !=0) // Mientras el buffer este
 lleno de datos se queda en while
210.          {
211.              controlReply=Serial1.readStringUntil('\r');// Lee
 la respuesta del modulo SIM900
212.              delay(25);
213.              controlReply.trim();
214.              controlReply1=controlReply;
215.              controlReply1.trim();
216.              Serial.println(controlReply1);
217.              if(controlReply1.equals(x)) // Compara la respuesta
 con controlEnviar="ERROR" incrementa una bandera
218.              {
219.                  Serial.println(y);
220.                  ErrorReply++;
221.                  if(ErrorReply==2)
222.                  {
223.                      ErrorReply=0;
224.                      i=1;
225.                  }
226.              }
```

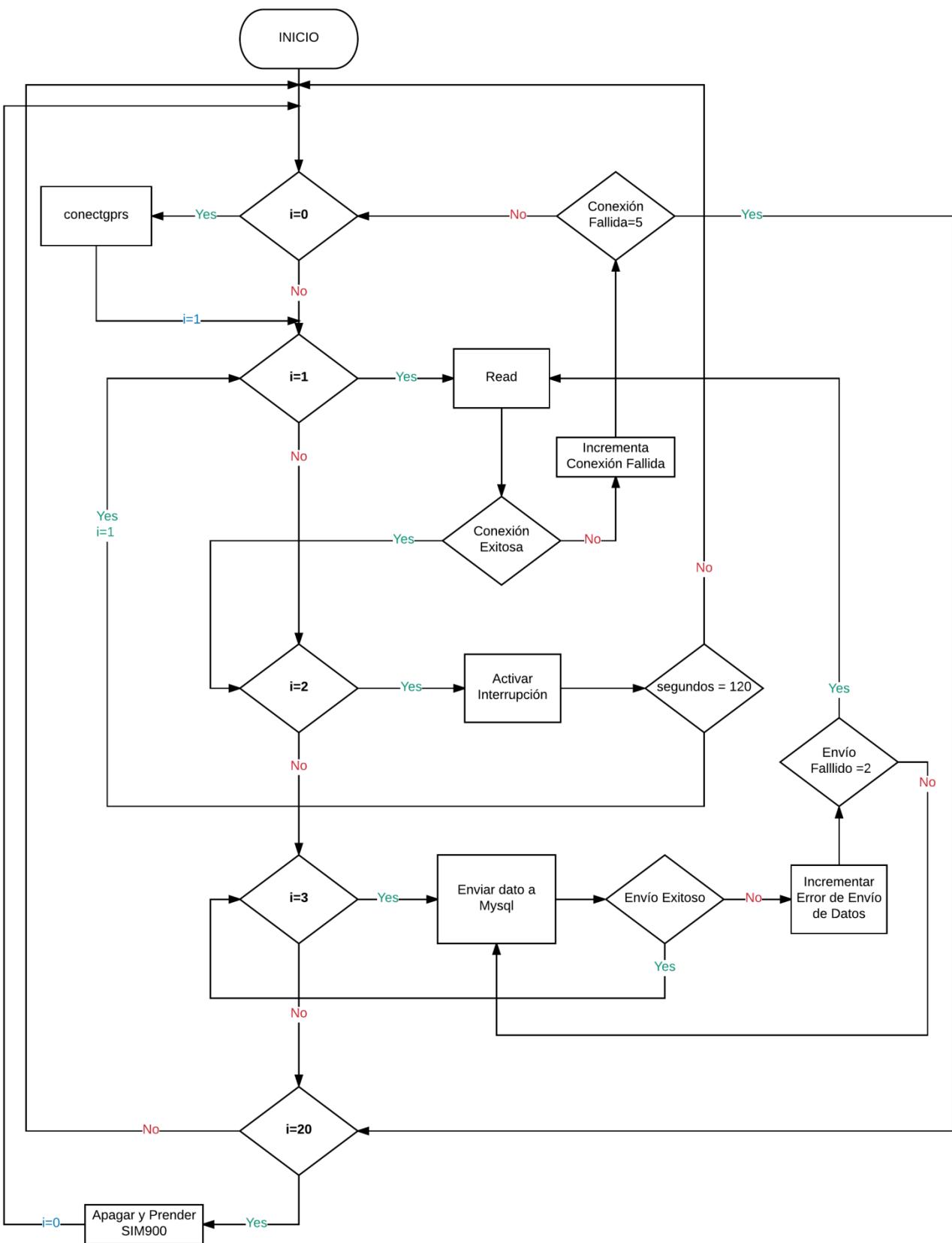
```
227.          }
228.      controlReply="";
229.      controlReply1="";
230.
231.  }
```

En el código se puede observar que se toman decisiones con respecto a la bandera **i**, es decir, dependiendo del estado de la bandera el código principal procederá a ejecutar las funciones, en este mismo sentido, hay otras bandera que indica cuantas veces la conexión falló para apagar y prender el **GPRS SIM900**. A continuación se explica cada estado de la bandera.

1. **i=0:** Se ejecuta la función **conectgprs**, dentro de ella se configuran los parámetros a través de los comandos **AT** para conectarse al servidor y enviar la petición de conexión.
2. **i=1:** Se ejecuta la función **read** para verificar la conexión al servidor, si la conexión es exitosa coloca **i=2** para activar la interrupción y continuar con él envío de datos, si no es exitosa, coloca **i=0** para volver intentar la conexión nuevamente con el servidor.
3. **i=2:** Activa la interrupción para controlar la verificación de conexión cada 2 minutos.
4. **i=3:** Envía datos al servidor **Mysql** de la entrada analógica “**0**”, al mismo tiempo, se verifica que el dato se haya enviado correctamente al servidor. Si el dato no fue enviado correctamente o la respuesta es “**ERROR**” se incrementa la bandera **ErrorReply** y si dicha bandera es igual a dos, se coloca **i=1** para verificar la conexión con el servidor.
5. **i=20:** Cabe destacar que, no se continuó la seguidilla de los números en la bandera **i**, ya que, a la hora de programar es más fácil identificar este caso, el cual, es apagar y encender el **GPRS SIM900**. Es decir, el programa entra en este caso cuando la conexión al servidor falla cinco veces o cuando la bandera **conexión** es igual a cinco (**conexión=5**). El incremento de esta bandera sucede dentro de la función **conectgprs**.

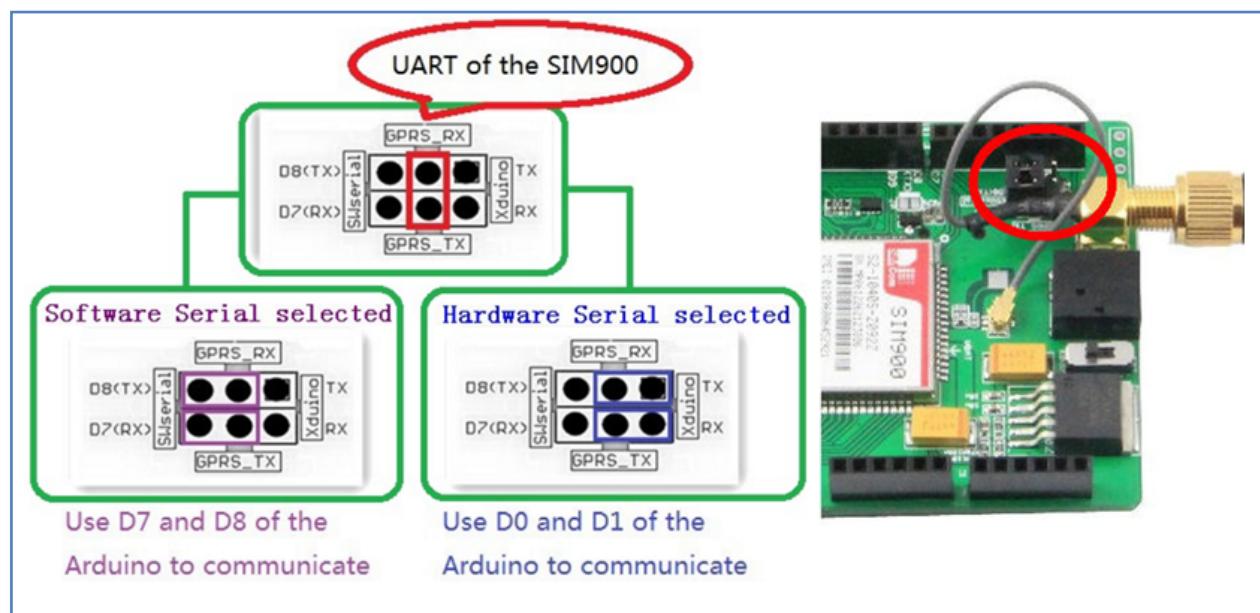
A continuación se muestra un diagrama de flujo para entender un poco mejor el **software**.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



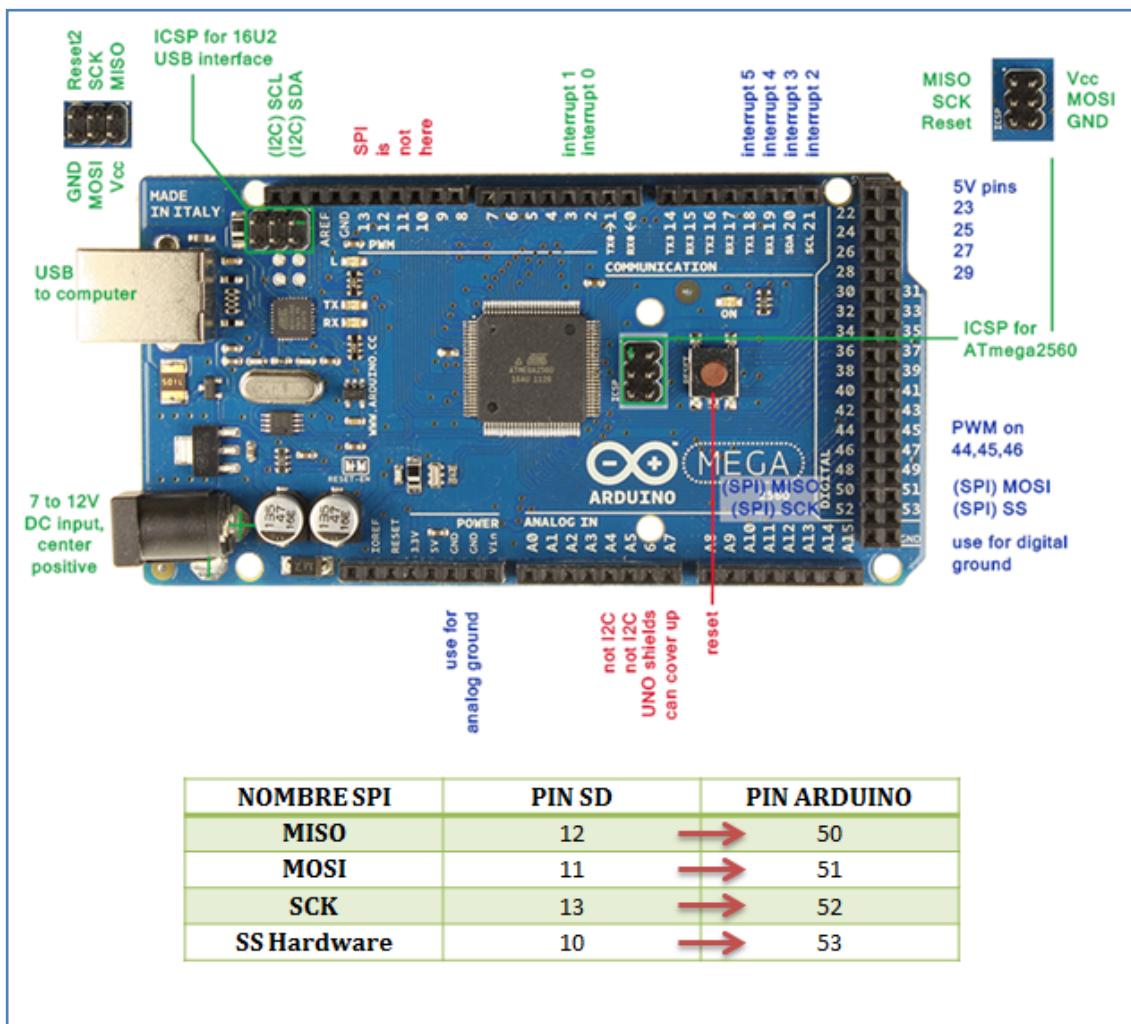
NOTA: se puede observar que, para esta rutina se utilizó el puerto **Serial uno** para comunicarse con el **GPRS SIM900** y el puerto **Serial cero (0)** se utilizó para enviar mensajes al monitor serial, con esto se tiene una mejor comunicación con el **GPRS SIM900** y se tiene aislado de los mensajes que se muestra en el monitor serial.

Cabe destacar que, se tiene que realizar un cableado desde el puerto **Serial uno** del **Ardunio Mega 2560** a los pines **7 (RX)** y **8 (TX)** del **GPRS SIM900**, para esto, se debe mover los **jumperes** del **SIM900** para que este se pueda comunicar a través de los pines **7 (RX)** y **8 (TX)**, es decir, se usaría el puerto serial software del **SIM900**.



RUTINA GRABAR DATOS EN SD

Para no perder los datos en caso de que el **GPRS SIM900** no pueda conectarse al servidor, se añadió un módulo **SD DEEK ROBOT V1.0 (Data Logger)**, para que así al momento de perder la conexión con servidor se guarden los datos en la **SD**. Para poder comunicarse con el modulo **SD DEEK ROBOT V1.0** se utiliza el puerto **SPI**. El puerto **SPI** en el **Arduino Mega 2560** está ubicado en los pines **50,51, 52, 53** y los pines del puerto **SPI** del módulo **SD** están ubicado en los pines **10,11,12** y **13**, por ende, se debe cablear el **arduino** con el modulo **SD** de la siguiente manera:



A continuación, se mostrara una pequeña rutina para grabar en la **SD**, esta rutina se insertara como función en el código principal del proyecto.

```
1. // Librerias SD:
2. #include <SPI.h>
3. #include <SD.h>
4. //Definimos el chipSelect, como es el Arduino Mega2560 se
   recomienda definirlo de la siguiente manera
5. const int chipSelect = (10,11,12,13); // Definir el chipSelect
   con los pines del SPI
6. int i=0;
7. File dataFile; // Definir el data file para grabar en la SD
8. int temp;
9. void setup()
10. {
11.     Serial.begin(19200);
12.     //Detectando SD y Configurando SD///////////
13.     pinMode(53, OUTPUT);
14.     if(!SD.begin(chipSelect)) // Comprobar si la SD Esta
   insertada o el cableado esta correctamente realizado
15.     {
16.         Serial.println("Fallo en tarjeta SD, Verificar
   Cableado o Tarjeta SD");
17.     }else
18.     {
19.         Serial.println("Tarjera SD OK");
20.     }
21.     dataFile=SD.open("data.txt",FILE_WRITE); // Abrir el
   archivo data.txt en modo escritura
22.     if (! dataFile) {
23.         Serial.println("Error al abrir el File");
24.         // Wait forever since we cant write data
25.         while (1) ;
26.     }
27.     /////////////
28. }
29.
30. void loop()
31. {
32.     temp = analogRead(0);
33.     dataFile.println(temp); // Llamar el dataFile
   anteriormente abierto para scribir la variable temp
34.     dataFile.flush(); // Se tiene que forzar el vaciado del
   buffer Serial para que escriba todos los datos
35.     Serial.println(temp);
36.     delay(500);
37.     //dataFile.close(); // Cada vez que se cierra el File se
   tiene que abrir nuevamente para poder escribir nuevamente
38. }
```

En el código se define las librerías **SPI** y **SD**, se declara, **chipselect** con los pines del puerto **SPI**, seguidamente, se declara como **file** la variable **datafile**, esto servirá para crear archivos **TXT**. Cabe destacar que, se tiene que definir como salida el pin **53** del **Arduino Mega 2560**, ya que, este es el **chipselect** o el **SS** tal como se indica en la tabla mostrada anteriormente.

Nota: Luego de escribir en el file o en el archivo, dicho archivo se cierra para tener la posibilidad de abrir otro archivo si así lo deseamos. Sin embargo, para este ejemplo no se cierra, ya que, se tendría que abrir nuevamente tal como está en la línea **21** del código, es decir, se tendría que colocar dentro del **loop** el “**SD.open**”.

El “**dataFile.flush**” se encarga de forzar el vaciado del buffer en la escritura de los datos.

RUTINA PARA LEER EL RELOJ DE TIEMPO REAL DEL MODULO SD

El reloj de tiempo real nos permitirá leer la hora, fecha y año, esto será de utilidad para guardar los datos en la SD e indicar la hora, fecha y año del dato guardado en ese instante. Cabe destacar que, el **Arduino** tiene la librería **RTClib.h** que facilita la programación del reloj de tiempo real (**RTC**).

Por otro lado, para comunicarse con el **RTC** del módulo **SD**, esta se realiza a través del puerto **I2C**. El puerto **I2C** del **Arduino Mega 2560** está en los pines **20** y **21**, esto va conectado a los pines **4** y **5** del módulo **SD**. Es importante comentar que, el cableado de la **SIM900** y del módulo **SD** se realizó a través de cables jumper, ya que, el módulo **SIM900** interfiere con el puerto **I2C** en estos pines.

El cableado del puerto **I2C** estaría del siguiente modo.

| NOMBRE DEL PIN | PIN ARDUINO | PIN MODULO SD |
|----------------|-------------|---------------|
| SDA | 20 | A4(4) |
| SCL | 21 | A5(5) |

A continuación se mostrara una pequeña rutina para guardar los datos del puerto analógico con su respectiva hora, fecha y año.

```
1. #include <Wire.h> // Puerto I2C para el RTC del modulo SD
2. #include <SPI.h> // Puerto SPI Para guardar datos en la SD
3. #include <SD.h> // Libreria SD
4. #include "RTClib.h" // Libreria RTC
5. RTC_DS1307 RTC; // Definir RTC
6. File dataFile; // Definir el File para guardar en la SD
7. const int chipSelect = (10,11,12,13); //Definimos el chipSelect,
   como es el Arduino Mega2560 se recomienda definirlo de la
   siguiente manera
8. int i=0;
9. void setup()
10. {
11.     Serial.begin(19200);
12.     Wire.begin();
13.     RTC.begin();
14.     //////////////////Configurando
15.     SD///////////////////////////////
16.     pinMode(53, OUTPUT); // Colocar como salida pin 53
17.     verificarsd(); // Verificar SD
18.     //////////////////Configurando RTC/////////////////////
19.     if (! RTC.isrunning())
20.     {
21.         Serial.println("RTC is NOT running!");
22.         // Le asigna hora y fecha al RTC con la ultima
23.         // compilacion del SKETCH se recomienda volcar el programa de una
24.         // vez al arduino
25.         // Es decir se carga una sola vez y luego se comenta
26.         // RTC.adjust(DateTime(__DATE__, __TIME__));
27.     }
28. }
29. void loop()
30. {
31.     DateTime now = RTC.now();
32.     i++;
33.     dataFile=SD.open("TESTRTC.txt",FILE_WRITE); // Abrir
   archivo TESTRTC.TXT en modo escritura
34.     Serial.print("Registrando en ");
35.     Serial.println("Prueba_RTC.txt");
36.     Serial.print(now.year(), DEC);
37.     Serial.print('/');
38.     Serial.print(now.month(), DEC);
39.     Serial.print('/');
40.     Serial.print(now.day(), DEC);
41.     Serial.print(' ');
42.     Serial.print(now.hour(), DEC);
43.     Serial.print(':');
44.     Serial.print(now.minute(), DEC);
45.     Serial.print(':');
46.     Serial.print(now.second(), DEC);
47.     Serial.print(" Temp: ");
48.     Serial.println(i);
49.     /////////////////////////////////
   dataFile.print(now.year(), DEC); // Escribir en el archivo
```

```
50.      dataFile.print('/'); // Escribir en el archivo
51.      dataFile.print(now.month(), DEC);
52.      dataFile.print('/');
53.      dataFile.print(now.day(), DEC);
54.      dataFile.print(' ');
55.      dataFile.print(now.hour(), DEC);
56.      dataFile.print(':');
57.      dataFile.print(now.minute(), DEC);
58.      dataFile.print(':');
59.      dataFile.print(now.second(), DEC);
60.      dataFile.print(" Temp: ");
61.      dataFile.println(i);
62.      dataFile.flush();
63.      dataFile.close();
64.      delay(1000);
65.  }
66. void verificarsd()
67. {
68.     if (!SD.begin(chipSelect))
69.     {
70.         Serial.println("Fallo en tarjeta SD, Verificar Cableado
o Tarjeta SD");
71.     }else
72.     {
73.         Serial.println("Tarjeta SD OK");
74.     }
75. }
76. }
77. }
```

En el código se puede observar cómo se inicia el puerto **I2C** con la función “**Wire.h**” y seguidamente se inicia la función “**RTClib.h**”. En el “**void setup**” se observa que se configura el **RTC**, sin embargo, la línea 23 de código esta comentada, ya que, esta se utiliza para configurar la hora, fecha y año actual. La librería toma la hora, fecha y año de la última compilación del programa, por ende, se recomienda cargar el programa inmediatamente al **arduino**.

En el “**void loop**” se llama a la función del **RTC** con **RTC.now** y se guarda en la variable **now** para luego mostrarla en el monitor serial y guardarla en el archivo.

- **Now.year:** Se obtiene el año.
- **Now.month:** Se obtiene el mes.
- **Now.day:** Se obtiene el día.
- **Now.hour:** Se obtiene la hora.
- **Now.minute:** Se obtiene los minutos.
- **Now.second:** Se obtiene los segundos.

RUTINA DE ENVIAR DATOS A MYSQL Y GUARDAR EN TARJETA SD EN CASO DE FALLAR LA CONEXIÓN AL SERVIDIRO CON SUS RESPECTIVA FECHA, HORA Y AÑO EN EL QUE SE GUARDO EL DATO

A continuación, se mostrara el código que comprende todas las funciones descritas anteriormente, dicho código se realizó en base a las rutinas explicadas anteriormente.

En este código se toman acciones en caso de que la conexión al servidor falle, incluso reiniciando el **GPRS SIM900** ([apagar y prender SIM900](#)). Es decir, cuando la conexión falla por segunda vez al servidor, se reinicia el **GPRS SIM900**, sin embargo, existe la posibilidad de que el servidor este fuera de servicio o la señal no esté en su estado óptimo. Por ende, se toma acciones de grabar los datos en la **SD** con su respectiva fecha, hora y año en caso de que el **GPRS SIM900** sea reiniciado por segundas vez.

El **void loop** o el código principal se maneja con la bandera **i** como se explica a continuación:

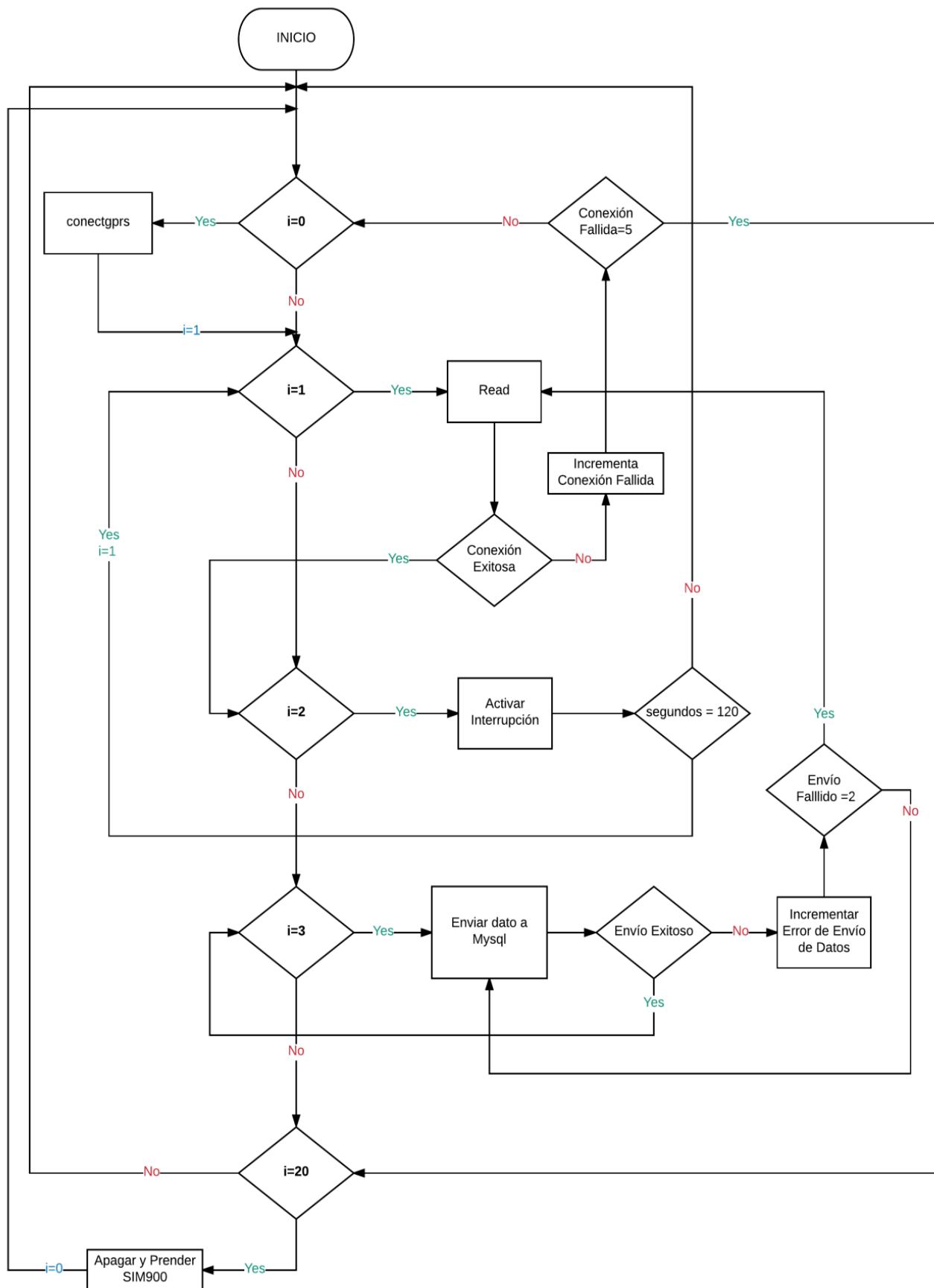
1. **i=0:** Se ejecuta la función **conectgprs**, dentro de ella se configuran los parámetros a través de los comandos **AT** para conectarse al servidor y enviar la petición de conexión.
2. **i=1:** Se ejecuta la función **read** para verificar la conexión al servidor, si la conexión es exitosa coloca **i=2** para activar la interrupción y continuar con él envío de datos, si no es exitosa, coloca **i=0** para volver intentar la conexión nuevamente con el servidor.
3. **i=2:** Activa la interrupción para controlar la verificación de conexión cada 2 minutos.
4. **i=3:** Envía datos al servidor **Mysql** de la entrada analógica “**0**”, al mismo tiempo, se verifica que el dato se haya enviado correctamente al servidor. Si el dato no fue enviado correctamente o la respuesta es “**ERROR**” se incrementa la bandera **ErrorReply** y si dicha bandera es igual a dos se coloca **i=1** para verificar la conexión con el servidor.
5. **i=20:** Cabe destacar que, no se continuó la seguidilla de los números en la bandera **i**, ya que, a la hora de programar es más fácil identificar este caso, el cual, es apagar y encender el **GPRS SIM900**. Es decir, el programa entra en este caso cuando la conexión al servidor falla cinco veces o cuando la bandera **conexión** es

igual a cinco (**conexión=5**). El incremento de esta bandera sucede dentro de la función **conectgprs**.

6. **i=30:** Cabe destacar que, no se continuó la seguidilla de los números en la bandera **i**, ya que, a la hora de programar es más fácil identificar este caso, el cual es, grabar los datos en la tarjeta de memoria **SD**. Es decir, el programa entra en este caso cuando se reinicia el **GPRS SIM900** mas de dos veces o cuando la bandera “**Reinicios**” es igual a dos (**Reinicios=2**). El incremento de dicha bandera se realiza cuando se reinicia el **GPRS SIM900**.

A continuación se muestra un diagrama de flujo para comprender un poco mejor el **software**.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)



Mauricio Verdugo

A continuación se muestra el código descrito anteriormente.

```
1. // En esta version se usa el puerto serial 1 para comunicarse
   con el sim900
2. /////////////////////////////////////////////////////////////////////
3. int i=0,mostrar=0,e=0,x=0,segundos=0,segundosSD=0,ia=0,conexion=
   0,ErrorReply=0,NumReinicios=0;
4. float k=0;
5. int addr=0, Reinicios=0,DatosEnSD=0;
6. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
7. int Reply=0;
8. // Librerias SD Y RTC/////////////////////////////
9. #include <SPI.h>
10.    #include <SD.h>
11.    #include <Wire.h> // Iniciar I2C
12.    #include "RTClib.h" // Libreria para el RTC
13.    RTC_DS1307 RTC;
14.    const int chipSelect = (10,11,12,13); //Definimos el
   chipSelect, como es el Arduino Mega2560 se recomienda definirlo
   de la siguiente manera
15.    File dataFile;
16.    /////////////////////////////////
17.    void setup()
18.    {
19.        i=0;
20.        RTC.begin(); //Iniciando RTC
21.        Wire.begin(); // Iniciando I2C
22.        Serial.begin(19200);
23.        Serial1.begin(19200);
24.        // onGPRS();
25.        delay(10000); //tiempo para ingresar a la red telefonica
26.        Serial.println("Serial ON\r\n");
27.        delay (100);
28.        ////////////////////CONFIGURANDO
   INTERRUPCIONES/////////////////////////////
29.        cli(); // Deshabilitar interrupciones globales
30.        TCCR4A=0; //Iniciando Registro
31.        TCCR4B=0; //Iniciando Registro
32.        OCR4A=15624; // Valor el cual timer se va a comparar con
   el CTC para reiniciar en 1 Seg
33.        TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
34.        TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
   con el CS42
35.        TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
   con el CS40
36.        sei(); // Activar las interrupciones
37.        //////////////////Configurando
   SD/////////////////////////////
38.        pinMode(53, OUTPUT);
39.        verificarsd();
40.        //////////////////Configurando RTC/////////////////////////////
41.        if (!RTC.isrunning())
42.        {
43.            Serial.println("RTC ERROR");
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
44.          // Le asigna hora y fecha al RTC con la ultima
        compilacion del SKETCH se recomienda volcar el programa de una
        vez al arduino
45.          // Es decir se carga una sola vez y luego se comenta
46.          //RTC.adjust(DateTime(__DATE__, __TIME__));
47.      }
48.
49.  }
50. ///////////////////////////////////////////////////////////////////
51. void loop()
52. {
53.     Serial.flush();
54.     Serial1.flush();
55.     if(i==0)
56.     {
57.         conectgprs(); // Funcion conectar GPRS se ejecuta solo
        cuando i=0
58.     }
59.     if (i==1)
60.     {
61.         Read(); //Funcion Read se ejecuta cuando i=1
62.     }
63.     if(i==2)
64.     {
65.         segundos=0;
66.         TIMSK4 =(1<<OCIE4A); // activando interrupcion por
        comparacion
67.         i=3;
68.     }
69.     if (i==3)
70.     {
71.         enviardatoAnalog(0); // Llama a la funcion enviar
        dato analogico, AIO
72.     }
73.     if(i==20)
74.     {
75.         offGPRS();
76.         delay(3000);
77.         onGPRS();
78.         if(Reinicios==2) // Verificar el numero de reinicios
        para asi grabar en la SD
79.     }
80.     i=30; // colocar la bandera 30, esto indica
        dentro el loop principal que se iniciar la funcion guardar en
        tarjeta sd.
81.     Reinicios=0;
82.     Serial.println("Registrando datos en tarjeta
        SD"); // Registrando datos en la tarjeta SD
83.     TIMSK4 =(1<<OCIE4A); // activando interrupcion
        por comparacion, para verificar luego de 2 minutos si hay
        conexion con el servidor.
84.     }else
85.     i=0;
86. }
87. if(i==30)
88. {
89.     if(DatosEnSD==1)
90.     {
91.         k=LeerAnalog(0);
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
92.           sdw("Temp.txt"," Temp: ",k);
93.           DatosEnSD=0;
94.       }
95.   }
96.
97.
98.   }
99.   ////////////////FUNCION CONECTAR A
100.  GPRS///////////////
101. void conectgprs()
102. {
103.     if(Serial.available()==0) // Verificar Serial disponible
104.     {
105.         TIMSK4 = (0<<OCIE4A); // Desactivar interrupcion
106.         Serial.println("Conectando..");
107.         delay(300);
108.         Serial1.print("AT+CGATT?\r\n"); // Comando AT para entrar
109.         a conexion TCP
110.         clearBuffer();
111.         delay(1000);
112.         Serial1.print("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\"\r\n"); // Configurar APN, USUARIO y CONTRASEÑA
113.         clearBuffer();
114.         delay(1000);
115.         Serial1.print("AT+CIICR\r\n"); // Levantar conexion GPRS
116.         clearBuffer();
117.         delay(1000);
118.         Serial1.print("AT+CIFSR\r\n"); // Obtener direccion IP
119.         clearBuffer();
120.         delay(300);
121.         clearBuffer();
122.         i=1;
123.     }
124. }
125. }
126. /////////////////
127. ///////////////LIMPIAR
128. BUFFER/////////////
129. void clearBuffer()
130. {
131.
132.     while (Serial1.available() !=0)
133.     {
134.         Serial.println(Serial1.readString()); //Usado para cuando
135.         se quiere leer a respuesta del sim900, para ello, se tiene que
136.         comentar las 3 lineas siguientes y descomentar esta linea.
137.         Clear=Serial1.readStringUntil('\n'); // Limpiando Buffer,
138.         Guardando respuesta en la variable Clear
139.         Clear=""; // Iniciar variable en 0
140.     }
141. ///////////////VERIFICACION DE
142. CONEXION/////////////
```

```
141.     void Read()
142.     {
143.         Serial.println("Verificando Conexion");
144.         if(Serial1.available()==0)
145.         {
146.             TIMSK4 =(0<<OCIE4A); // Desactivar interrupcion
147.             Serial1.println("AT+CIPSTATUS\r\n"); // Envia el comando
Status para verificar conexion
148.             delay(100);
149.             while (Serial1.available()!=0)// Mientras el buffer
este lleno de datos se queda en while
150.             {
151.                 control=Serial1.readStringUntil('\n');// Lee la
respuesta del modulo SIM900
152.                 delay(25);
153.                 control1=control; // Pasamos la variable control a
control1 para luego compararla
154.                 control.trim(); // Corta los espacio del string
155.                 control1.trim();
156.                 Serial.println(control);
157.             }
158.             if(control1.equals(control2)) // Compara la
respuesta con control2 = "STATE: CONNECT OK"
159.             {
160.                 Serial.println("Conexion Exitosa");
161.                 i=2;
162.                 conexion=0;
163.             } else
164.             {
165.                 Serial.println("Conexion Fallida");
166.                 i=0;
167.                 conexion++;
168.             }
169.         }
170.         if (conexion==5)
171.         {
172.             i=20;
173.         }
174.         clearBuffer();
175.         control=""; // Iniciar la variable en 0 para la proxima
comparacion
176.         control1="";
177.     }
178.     /////////////////////////////////
179.     //////////////////Funcion Interrumpir para verificar
Status/////////////////
180.     ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
funcion cuando la comparacion del CTC y el timer es 15624
181.     {
182.         segundos++;
183.         segundosSD++;
184.         if(segundos==120) // Entra a la condicion luego de 2
minutos.
185.         {
186.             i=1;
187.             segundos=0;
188.         }
189.         if(segundosSD==10)
190.         {
```

```
191.     DatosEnSD=1;
192.     segundosSD=0;
193. }
194.
195. }
196. ///////////////////////////////////////////////////////////////////
197. ///////////////////////////////////////////////////////////////////Enviar Datos
198. ///////////////////////////////////////////////////////////////////Analogicos
199. void enviardatoAnalog(int y)
200. {
201.     if(Serial1.available()==0)
202.         x=analogRead(y); //Se lee el puerto analogico a traves
203.         //por paso de parametro de la funcion en este caso el puerto
204.         //analogico 0
205.         Serial1.println("AT+CIPSEND\r\n"); // Preparar el mensaje
206.         a enviar
207.         delay(1000);
208.         Serial1.print("GET
209.         /arduino/registros.php?temperatura="); // Escribir mensaje a
210.         enviar al servidor mediante el metodo GET, se envia sin fin de
211.         linea para poder enviar el dato "x", luego de enviar el dato se
212.         envia fin de linea y retorno de carro "r\n"
213.         delay(300);
214.         Serial1.print(x);
215.         delay(300);
216.         Serial1.print(" HTTP/1.1\r\n"); // Se envia la version
217.         del protocolo, para este caso HTTP/1.1
218.         delay(1000);
219.         Serial1.print("Host: 190.215.92.114\r\n"); // De acuerdo
220.         al protocolo HTTP/1.1 en la cabecera debe especificar el host
221.         receptor
222.         delay(1000);
223.         Serial1.print("Connection: Keep-Alive"); // El protocolo
224.         HTTP/1.1 permite conexiones persistente, es decir, la conexion
225.         queda abierta durante el tiempo programado en el servidor. Pues,
226.         ya que el servidor responde a la peticion y cierra la
227.         conexion.Sin embargo, esta configuracion se realizo en el
228.         servidor y se puede omitir esta linea de codigo
229.         delay(1000);
230.         Serial1.println("\r\n"); // Entre linea
231.         //delay(1000);
232.         Serial1.println("\xA1"); // Caracter especial para enviar
233.         mensaje al servidor
234.         Reply=WaitReply("ERROR","Envio Fallido","Connection:
235.         close","Cerrando Conexion"); // Se pasan los parametros a la
236.         funcion para verificar la respuesta, esta retorna 1 o 2
237.         if(Reply==1) // Si el retorno es 1 significa que hay un
238.         ERROR en el envio
239.         {
240.             ErrorReply++; // Si hay un error en el envio se
241.             incrementa ErrorReply
242.             if(ErrorReply==2) // Si hay dos errores en el envio se
243.             coloca la bandera i=1, para chequear la conexion con el servidor
244.             {
245.                 ErrorReply=0;
246.                 i=1;
247.             }
248.         }
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
228.         if(Reply==2) // Si la respuesta en Reply retorna "2",
    esto quiere decir que alcanzo el limite de datos enviados y por
    ende se cierra la conexion, en esto influye el keepAlive
    configurado en el servidor
229.         {
230.             //Serial1.println("AT+CIPCLOSE"); //se cierra la
    conexion, ya que, la conexion abierta solo permite 100 envios de
    datos, almenos que se modifique este parametro en el servidor.
231.             i=0;
232.         }
233.         //clearBuffer();
234.         //delay(300);
235.         //clearBuffer();
236.     }
237. }
238. ///////////////////////////////////////////////////////////////////
239. ////////////////////////////////////////////////////////////////// APAGAR GPRS EN CASO DE FALLAR
    CONEXION//////////////////////////////////////////////////////////////////
240. void offGPRS()
241. {
242.
243.     delay(3000);
244.     clearBuffer();
245.     if(Serial1.available()==0)
246.     {
247.         Serial.println("Reiniciando SIM900...");
248.         Serial.println("AT+CPWD=1");
249.         delay(100);
250.         clearBuffer();
251.         Reinicios++; // Bandera para comparar en la condicion
    i=30
252.         NumReinicios++; // Para indicar en la SD cuantas
    veces se ha reiniciado
253.         sdw("Reset.txt"," Sim900 Reiniciado
    #",NumReinicios); // Se pasa el parametro "nombre del
    archivo","Lo que se desea escribir"," Variable que se quiera
    reflejar"
254.
255.     }
256.
257. }
258. void onGPRS()
259. {
260.     Serial.println("Encendiendo SIM900...");
261.     delay(3000);
262.     pinMode (9,OUTPUT);
263.     digitalWrite (9, LOW);
264.     delay(1000);
265.     digitalWrite (9, HIGH);
266.     delay(2000);
267.     digitalWrite (9,LOW);
268.     delay(3000);
269.     conexion=0;
270.     delay(3000);
271.     clearBuffer();
272. }
273. ///////////////////////////////////////////////////////////////////
274. ////////////////////////////////////////////////////////////////// FUNCION LEER RESPUESTA////////////////////////////////////////////////////////////////
```

```
275.
276.     int WaitReply(String x, String y, String z, String w)
277.     {
278.         int r=0;
279.         String controlReply;
280.         String controlReply1;
281.         while (Serial1.available() != 0) // Mientras el buffer este
282.             lleno de datos se queda en while
283.             {
284.                 controlReply=Serial1.readStringUntil('\r'); // Lee
285.                 la respuesta del modulo SIM900
286.                 delay(25);
287.                 controlReply.trim();
288.                 controlReply1=controlReply;
289.                 controlReply1.trim();
290.                 Serial.println(controlReply1);
291.                 if(controlReply1.equals(x)) // Compara la
292.                     respuesta con controlReply
293.                     {
294.                         Serial.println(y);
295.                         r=1;
296.                     }
297.                 if(controlReply1.equals(z)) // Compara la respuesta
298.                     con controlReply
299.                     {
300.                         Serial.println(w);
301.                         r=2;
302.                     }
303.                 }
304.
305.     ///////////Funcion Grabar en
306.     SD///////////////
307.     void sdw(char filename[], String z, int x) // PASO EL
308.         PARAMETRO DEL FILE
309.         {
310.             DateTime now = RTC.now();
311.             dataFile=SD.open(filename,FILE_WRITE);
312.             Serial.print("Registrando en ");
313.             Serial.write(filename);
314.             Serial.println('\n');
315.             dataFile.print(z); // Referencie de lo que se vaa a
316.             guardar
317.             dataFile.print(x); // variable que se va a guardar
318.             dataFile.print(' ');
319.             dataFile.print(now.year(), DEC);
320.             dataFile.print('/');
321.             dataFile.print(now.month(), DEC);
322.             dataFile.print('/');
323.             dataFile.print(now.day(), DEC);
324.             dataFile.print(' ');
325.             dataFile.print(now.hour(), DEC);
326.             dataFile.print(':');
327.             dataFile.print(now.minute(), DEC);
328.             dataFile.print(':');
329.             dataFile.println(now.second(), DEC);
330.             dataFile.flush();
```

```
328.         dataFile.close();
329.     }
330.     ///////////////////////////////////////////////////
331.     ///////////////////////////////////////////////////Funcion Verificar
332.     void verificarsd()
333.     {
334.         if (!SD.begin(chipSelect))
335.         {
336.             Serial.println("Fallo en tarjeta SD, Verificar Cableado
o Tarjeta SD");
337.         }
338.         }else
339.         {
340.             Serial.println("Tarjeta SD OK");
341.         }
342.     }
343. }
344. ///////////////////////////////////////////////////
345. ///////////////////////////////////////////////////Leer Dato
Analogico/////////////////////////////////////////////////
346. float LeerAnalog(int y)
347. {
348.     float x;
349.     x=analogRead(y);
350.     return x;
351. }
```

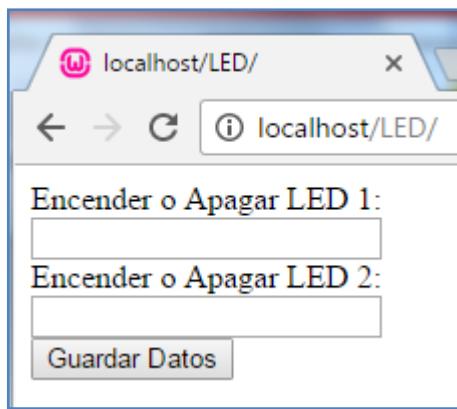
Nota: En este código se usa el puerto **Serial 1** para enviar los comandos **AT** y configurar el módulo **GPRS SIM900** y el puerto **Serial 0** para enviar mensaje al monitor serial.

La función **clearBuffer** además de limpiar el **Buffer**, muestra al usuario la respuesta que envía el **GPRS SIM900** cuando se envían los comandos **AT** al **GPRS SIM900**.

ENCENDER LED VIA WEB

Para encender un **LED** vía **WEB**, se debe realizar una programación en **PHP** en el servidor. Para ello, se realizara tres pequeñas rutinas en el servidor con **PHP**.

1. El primero código, es para que el usuario introduzca los estados de los LED, es decir, el usuario introducirá un “**1**” para encender el LED y un “**0**” para apagar el **LED**. El código consta de un formulario que se muestra al usuario para introducir los estados de los **LED**, el formulario está hecho en **HTML** dentro de **PHP** como se explicó en el apartado [conexión a base de datos mediante PHP](#). A continuación se muestra la plantilla para el usuario.



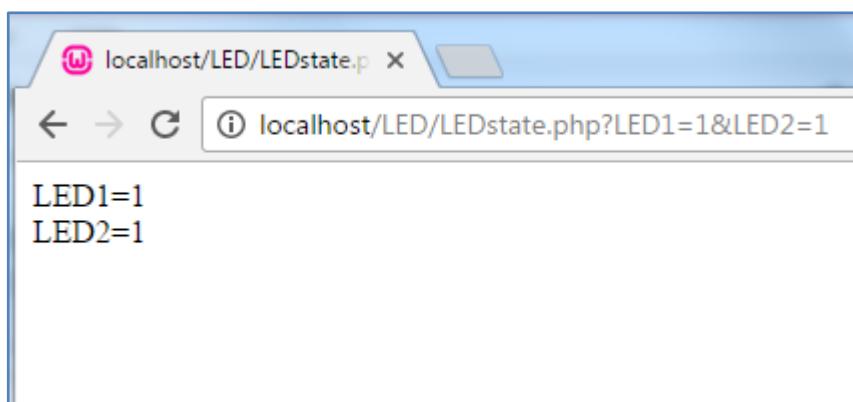
A continuación se muestra el código del formulario realizado en **HTML** dentro de **PHP**.

```
1.  
2. <?php
```

```
3.  
4. ?>  
5.  
6. <!--Se Llama el archivo LEDstate.php, el cual, tiene las  
    consulta de mysql  
7.           y la adquisición de datos se realiza a traves del metodo  
    GET, este formulario esta realizado bajo HTML -->  
8. <form action="LEDstate.php" method="get" name="form">  
9.   Encender o Apagar LED 1: <br/> <!--Se muestra en la pagina  
    Encender o apagar LED 1 y el br es para salto de linea -->  
10.  <input type = "stat" name="LED1"/> <br/> <!-- Entrada tipo  
    estado y lo guarda en la variables LED1 -->  
11.  Encender o Apagar LED 2: <br/>  
12.  <input type = "stat" name="LED2"/> <br/><!-- Entrada tipo  
    estado y lo guarda en la variables LED2 -->  
13.  <input type = "submit" value="Guardar Datos"/><!-- Entrada  
    tipo enviar (boton para enviar los datos), envia los datos al  
    archivo LEDstate.php -->  
14. </form>
```

En el código se puede observar que, se llama el archivo “**LEDstate.php**”, en donde este realiza la conexión con la base de datos (**BD**) y registra los valores introducidos por el usuario a la **BD**. Es importante destacar que el archivo “**LEDstate.php**” es el segundo código, el cual, explicaremos a continuación.

2. El segundo código se encarga de obtener los datos introducido por el usuario en el formulario, los datos se obtienen a través del método “**GET**”, seguidamente, se realiza la consulta a la base de datos y finalmente muestra el valor introducido por el usuario en el navegador o en la web. En la siguiente figura se puede apreciar como se muestra los datos introducidos por el usuario.



A continuación se muestra el código realizado en **PHP**.

```

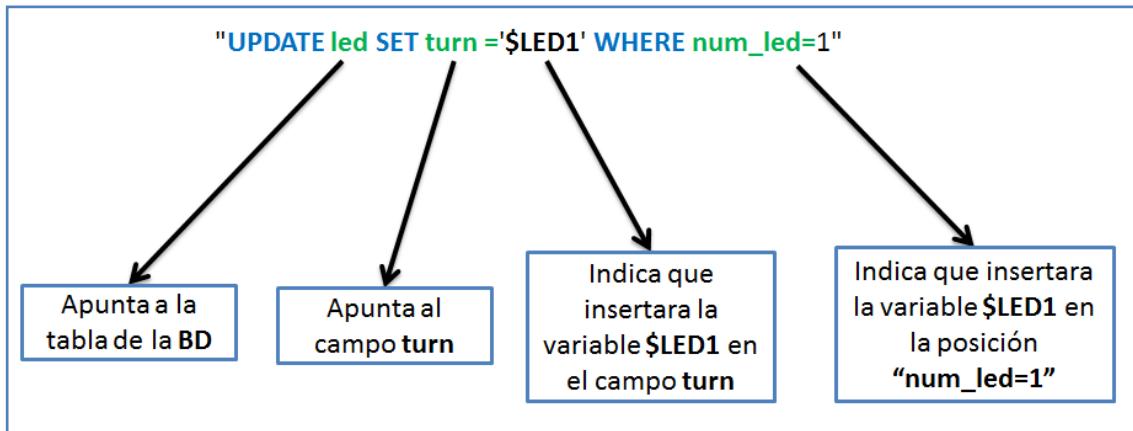
1. <?php
2. include 'conexion.php'; // Llama al archivo conexion mediante el
   include
3. $LED1 = $_GET["LED1"]; // Se definen las variables, cabe
   destacar que, las variables tienen que coincidir con el nombre
   de las variables definida en el formulario.
4. $LED2 = $_GET["LED2"];
5. $insertar1= "UPDATE led SET turn ='$LED1' WHERE num_led=1"; //
   Se define una consulta en la variable $insertar1 para actualizar
   el dato, en la tabla led y en el campo tabla turn, posicion 1
6. $insertar2= "UPDATE led SET turn ='$LED2' WHERE num_led=2"; // Se
   define una consulta en la variable $insertar2 para actualizar el
   dato, en la tabla led y en campo turn, posicion 2
7. $resultado1= mysqli_query ($conexion,$insertar1); // Se ejecuta
   la consulta definida en la variable insertar1
8. $resultado2= mysqli_query ($conexion,$insertar2);
9. if(!$resultado1)
10. {
11.     echo 'Error Al Cambiar Estado LED1<br/>';
12. }
13. else
14. {
15.     echo "LED1="; // se imprime en pantalla
   LED1= y el valor actualizado por el usuario
16.     echo "$LED1<br/>";
17. }
18. if(!$resultado2)
19. {
20.     echo 'Error Al Cambiar Estado LED2<br/>';
21. }
22. else
23. {
24.     echo "LED2="; // se imprime en pantalla
   LED2= y el valor actualizado por el usuario
25.     echo "$LED2<br/>";
26. }
27. //Cerrar conexion
28. mysqli_close($conexion);
29. $textfile="C:\wamp64\www\LED\STATE.txt"; // se define la
   ubicacion del archivo donde se va a escribir
30. $fileLocation="$textfile"; // se define en la variable
   $fileLocation
31. $fh=fopen($fileLocation, 'w'); // se define en la variabbles
   $fh la apertura del archivo en modo escritura (w)
32. $stringToWrite="$LED1,$LED2"; // se define en la variables
   $stringToWrite lo que se desea escribir
33. fwrite($fh,$stringToWrite); // se abre el archivo y se
   escribe lo asignado en la variables $stringToWrite
34. fclose($fh); // se cierra el archivo
35. //header("Location: http://localhost/LED/"); // Se utiliza
   para volver a una pagina o saltar a otra pagina
36. ?>

```

Primeramente en el código se obtiene las variables introducida por el usuario a través del método “**GET**”, luego, se realizan dos consultas separadas, dichas consultas se encargan de actualizar los datos introducido por el usuario por lo que están en la base de datos, es decir, sobre escribe en la base de datos. Finalmente se escriben los valores en un archivo

de texto “STATE.txt”, esto se realiza solo para corroborar la escritura de los datos introducidos a la **BD**.

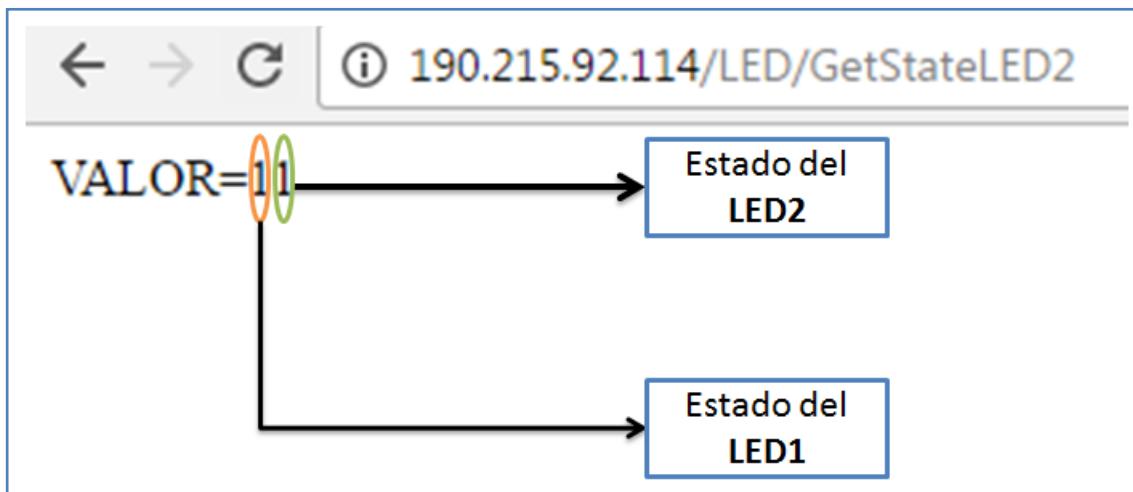
A la siguiente imagen se explica la estructura de la consulta realizada para el **LED1**.



En este código también es llamado o incluido el código conexión, la cual, nos permitirá conectar a la base de datos, dicho código esta detallado en el apartado [conexión a base de datos mediante PHP](#).

3. El tercer código se encarga de realizar una consulta a la base de datos para obtener los valores introducidos por el usuario, es decir, a esta dirección de archivo es donde apuntara el **Arduino** para que el servidor le responda o le entregue los valores de los **LED** que introdujo el usuario.

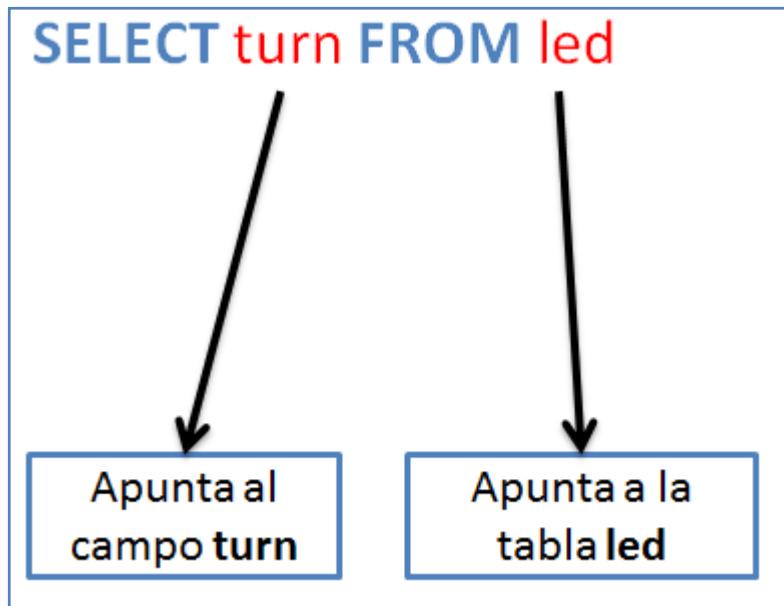
El **Arduino** apuntara a la siguiente dirección mediante petición **GET** “190.215.92.114/LED/GetStateLED2”, si colocamos dicha dirección en el navegador web nos arrojara el siguiente resultado.



A continuación, se muestra el código para realizar la consulta a la base de datos u obtener los valores introducido por el usuario para cada **LED**.

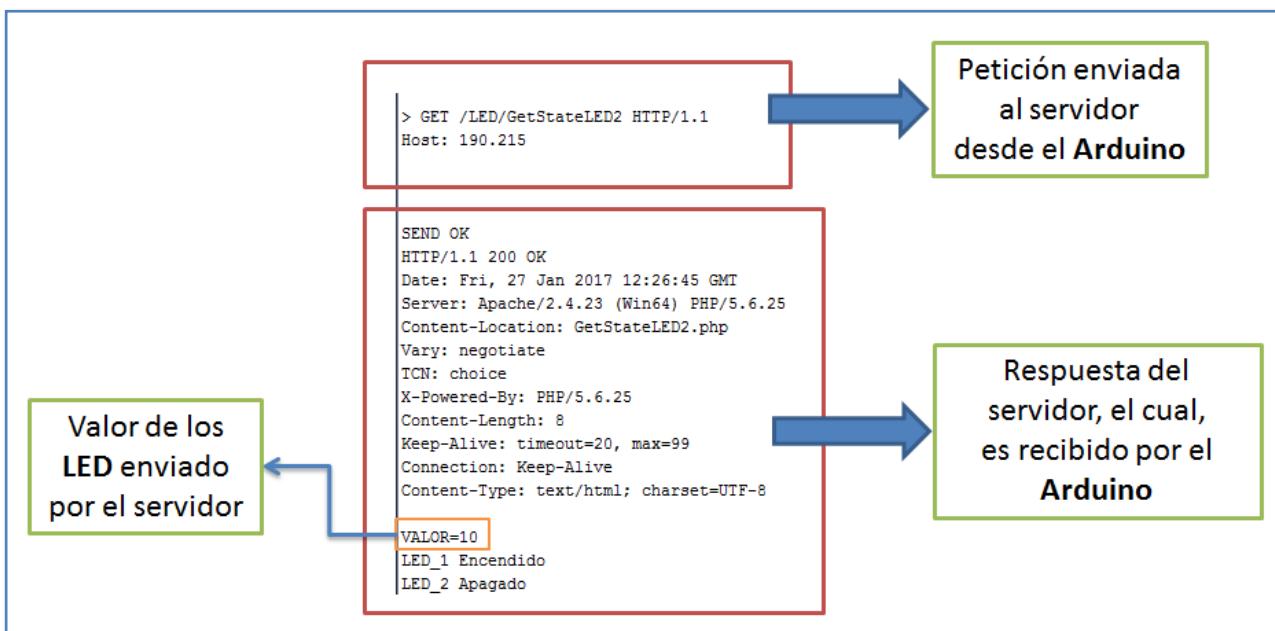
```
1. <?php
2. include 'conexion.php'; // Llama al archivo conexion mediante el
   include
3. $insertar="SELECT turn FROM led"; // Se define la consulta. La
   consulta es en la tabla led y en el campo turn
4. $resultado= mysqli_query ($conexion,$insertar);
5. echo 'VALOR=';
6. while($LED= mysqli_fetch_assoc($resultado)) // Realizar la
   consulta mediante un while, ya que, se necesita que retorne todo
   los volares del campo turn de la tabla led.
7. {
8. if (!$resultado)
9. {
10.           echo 'Error<br/>';
11.         }
12.     else
13.     {
14.
15.           echo $LED['turn'];//(las llamas por el nombre del
   campo q tengan en la bd)
16.
17.         }
18.     }
19.     mysqli_close($conexion);
20. ?>
```

La consulta que se realiza en la variable **\$insertar** indica que se seleccionara la tabla **led** y el campo **turn**. En la variable **\$resultado**, se define la consulta predefinida en **\$insertar**. A continuación se muestra la estructura de la consulta.



En la línea 6 del código se encuentra el ciclo **While** realizando la consulta “`($LED= mysqli_fetch_assoc($resultado))`”, con este ciclo se obtendrá todos los valores del campo **turn**.

Ahora bien, con los códigos descrito anteriormente se puede realizar la petición **GET** desde el **Arduino**, tal como se ha mencionado anteriormente, dicha petición tendrá una respuesta del servidor como se muestra en la siguiente imagen.



El servidor envía una respuesta “completa” al **Arduino** indicando algunas configuraciones, fecha, hora, archivo, etc. Sin embargo, la respuesta que se debe tomar en cuenta es el valor de los **LED** como se muestra en la figura anterior “**VALOR=10**”, donde el **LED1** está en estado **ON** y el **LED2** está en el estado **OFF**. El mensaje “**VALOR=10**” se trata dentro el código del **Arduino** para obtener los valores de cada uno de los **LED**. A continuación se muestra el código para encender o apagar los **LED** vía **WEB**.

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
1. ///////////////
2. int i=0,mostrar=0,e=0,x=1;
3. String control1,control,control2="STATE: CONNECT
   OK",Clear,show; // Se declara la variable control2 para comparar
   la conexión
4. void setup()
5. {
6. pinMode(4,OUTPUT);
7. pinMode(3,OUTPUT);
8. i=0;
9. Serial.begin(19200);
10.    Serial1.begin(19200);
11.    delay(4000); //tiempo para ingresar a la red telefónica
12. }
13. void loop()
14. {
15.     if(i==0)
16.     {
17.         conectgprs(); // Función conectar GPRS se ejecuta solo
   cuando i=0
18.     }
19.     if (i==1)
20.     {
21.         Read(); //Función Read se ejecuta cuando i=1
22.     }
23.     if(i==2)
24.     {
25.         recibirLED(4,3) ; // Paso las salidas digital a activar
26.     }
27. }
28. ////////////////////FUNCIÓN CONECTAR A
   GPRS///////////////////////////////
29. void conectgprs()
30. {
31.     if(Serial.available()==0) // Verificar Serial disponible
32.     {
33.         Serial.println("Conectando..");
34.         delay(300);
35.         Serial1.print("AT+CGATT?\r\n"); // Comando AT para entrar
   a conexion TCP
36.         clearBuffer();
37.         delay(1000);
38.         Serial1.print("AT+CSTT=\"web.tmovil.cl\",\"web\",\"web\"\r\n");
   // Configurar APN, USUARIO y CONTRASEÑA
39.         clearBuffer();
40.         delay(1000);
41.         Serial1.print("AT+CIICR\r\n"); // Levantar conexión GPRS
42.         clearBuffer();
43.         delay(1000);
44.         Serial1.print("AT+CIFSR\r\n"); // Obtener dirección IP
45.         clearBuffer();
46.         delay(1000);
47.         Serial1.print("AT+CIPSTART=\"TCP\",\"190.215.92.114\",80\r\n");
   // Conectarse al servidor mediante TCP
48.         clearBuffer();
49.         delay(300);
50.         clearBuffer();
51.         i=1;
52.     }
53. }
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
54.      ///////////////////////////////////////////////////
55.      ///////////////////////////////////////////////////
56.
57.      /////////////////////////////////////////////////// LIMPIAR
58.      void clearBuffer()
59.      {
60.
61.      while (Serial1.available() !=0)
62.      {
63.          Serial.println(Serial1.readString());
64.          Serial.flush();
65.          Clear=Serial1.readStringUntil('\n'); // Limpiando Buffer,
   Guardando respuesta en la variable Clear
66.          Clear=""; // Iniciar variable en 0
67.      }
68.  }
69. ///////////////////////////////////////////////////
// 70. /////////////////////////////////////////////////// VERIFICACION DE
CONEXION/////////////////////////////////////////////////
71. void Read()
72. {
73.     if(Serial.available()==0) // Espera el Serial Disponible
74.     {
75.         Serial1.println("AT+CIPSTATUS\r\n"); // Envia el
   comando Status para verificar conexión
76.         delay(100);
77.         delay(100);
78.         while (Serial1.available() !=0) // Mientras el buffer
   este lleno de datos se queda en while
79.         {
80.             control=Serial1.readStringUntil('\n');// Lee la
   respuesta del modulo SIM900
81.             delay(25);
82.             control1=control; // Pasamos la variable control a
   control1 para luego compararla
83.             control.trim(); // Corta los espacio del string
84.             control1.trim();
85.             Serial.println(control1);
86.         }
87.         if(control1.equals(control2)||control1=="ALREADY
CONNECT") // Compara la respuesta con control2 = "STATE: CONNECT
OK"
88.         {
89.             Serial.write("Conexion Exitosa");
90.             i=2;
91.         } else i=0;
92.
93.     }
94.     clearBuffer(); // Limpiar Buffer
95.     control=""; // Iniciar la variable en 0 para la proxima
   comparación
96.     control1="";
97. }
98. ///////////////////////////////////////////////////
// 99. void recibirLED(int LED1, int LED2)
100. {
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
101.     int posicion=0, longitud=0, posicionLED1=0, posicionLED2=0;
102.     String controlReply="";
103.     String controlReply1="";
104.     if(Serial1.available()==0)
105.     {
106.         Serial1.println("AT+CIPSEND\r\n"); // Preparar el mensaje
107.         delay(1000);
108.         Serial1.print("GET /LED/GetStateLED2"); // Escribir
109.         mensaje a enviar al servidor mediante el metodo GET, se envia
110.         sin fin de linea para poder enviar el dato "x", luego de enviar
111.         el dato se envia fin de linea y retorno de carro "\r\n"
112.         delay(300);
113.         Serial1.print(" HTTP/1.1\r\n"); // Se envia la version
114.         del protocolo, para este caso HTTP/1.1
115.         delay(1000);
116.         Serial1.print("Host: 190.215.92.114\r\n"); // De acuerdo
117.         al protocolo HTTP/1.1 en la cabecera debe especificar el host
118.         receptor
119.         delay(1000);
120.         Serial1.println("\r\n"); // Entre linea
121.         Serial1.println("\xA");
122.         while (Serial1.available()!=0) // Mientras el buffer este
123.             lleno de datos se queda en while
124.         {
125.             controlReply=Serial1.readStringUntil('\r');// Lee
126.             la respuesta del modulo SIM900
127.             delay(5);
128.             controlReply.trim();
129.             controlReply1=controlReply;
130.             controlReply1.trim();
131.             Serial.println(controlReply1);
132.             if(controlReply1.equals("ERROR") || controlReply1.eq
133.                 uals("SEND FAIL"))// Compara la respuesta con controlReply
134.             {
135.                 i=0;
136.             }
137.             longitud=controlReply1.length(); // Se obtiene la longitud
138.             del texto enviado por el servidor
139.             posicion=controlReply1.indexOf("VALOR="); // Se obtiene la
140.             posicion de VALOR=
141.             posicion=posicion+6; // Se le suma 6, ya que "VALOR=" tiene
142.             6 caracteres y solo queremos leer los valores arrojados.
143.             posicionLED1=posicion+1; // se obtiene la posicion del
144.             valor del LED1
145.             posicionLED2=posicion+2; // Se obtiene la posicion del
146.             valor del LED2
147.             if (controlReply1.substring(posicion, posicionLED1)==="1") //
148.                 a partir de posicion+6 en adelante hasta posicion LED1.
149.             {
```

```
140.          Serial.println("LED_1 Encendido");
141.          digitalWrite(LED1,HIGH);
142.      }else if(controlReply1.substring(posicion,posicionLED1)
143.      =="0")
144.      {
145.          Serial.print("LED_1 Apagado");
146.          digitalWrite(LED1,LOW);
147.      }
148.      if (controlReply1.substring(posicionLED1,posicionLED2)=="1") // A partir de posicion LED1 hasta posicion LED2
149.      {
150.          Serial.println("LED_2 Encendido");
151.          digitalWrite(LED2,HIGH);
152.      }else if(controlReply1.substring(posicionLED1,posicionLED2)
153.      =="0")
154.      {
155.          Serial.println("LED_2 Apagado");
156.          digitalWrite(LED2,LOW);
157.      }
158.      controlReply="";
159.      controlReply1="";
160.  }
```

El código básicamente conecta al servidor haciendo la verificación de conexión tal como se ha explicado anteriormente, seguidamente, envía la petición **GET** al servidor y espera la respuesta del servidor para tomar las acciones correspondientes con los **LED**. Ahora bien, la respuesta recibida será la que se mostró en la imagen anterior, en esta respuesta que envía el servidor al **Arduino** se obtienen datos del mismo (hora, fecha, etc.), pero, solo se tiene que tomar el valor de los **LED** que envía el servidor.

Esta respuesta “**VALOR=10**”, se tiene que buscar en todos los datos que envía el servidor, para ello, se realizó las siguientes líneas de código (**línea 133**).

```

132.
133.     longitud=controlReply1.length(); // Se obtiene la longitud
      del texto enviado por el servidor
134.     posicion=controlReply1.indexOf("VALOR="); // Se obtiene la
      posicion de VALOR=
135.     posicion=posicion+6; // Se le suma 6, ya que "VALOR=" tiene
      6 caracteres y solo queremos leer los valores arrojados.
136.     posicionLED1=posicion+1; // se obtiene la posicion del
      valor del LED1
137.     posicionLED2=posicion+2; // Se obtiene la posicion del
      valor del LED2
138.     if (controlReply1.substring(posicion, posicionLED1)=="1") //
      a partir de posicion+6 en adelante hasta posicion LED1.
139.     {
140.         Serial.println("LED_1 Encendido");
141.         digitalWrite(LED1,HIGH);
142.     }else if(controlReply1.substring(posicion, posicionLED1)
      =="0")
143.     {
144.         Serial.print("LED_1 Apagado");
145.         digitalWrite(LED1,LOW);
146.     }
147.     if (controlReply1.substring(posicionLED1, posicionLED2)==""
      1)// A partir de posicion LED1 hasta posicion LED2
148.     {
149.         Serial.println("LED_2 Encendido");
150.         digitalWrite(LED2,HIGH);
151.     }else if(controlReply1.substring(posicionLED1, posicionL
      ED2)=="0")
152.     {
153.         Serial.println("LED_2 Apagado");
154.         digitalWrite(LED2,LOW);
155.     }
156.
157. }
158. controlReply="";
159. controlReply1="";
160. }

```

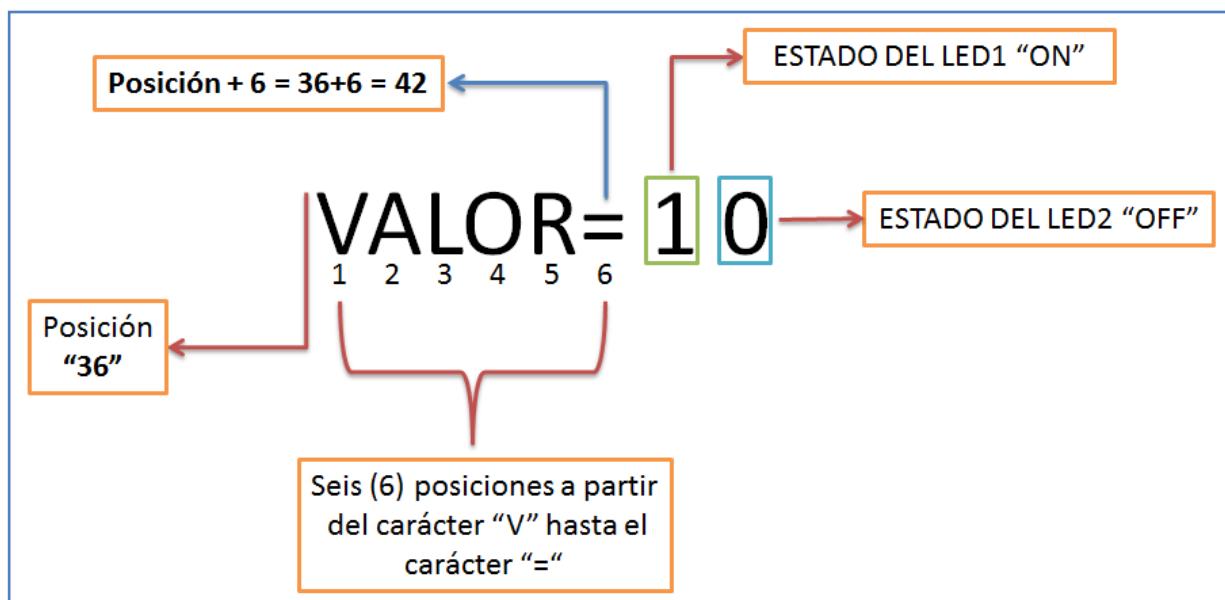
En la línea **133** del condigo se puede observar que se obtiene y se guarda la longitud del mensaje que envía el servidor, esto se realiza aplicando la función “**length()**” en la variable donde se obtiene le mensaje, es decir, en la variable **“controlReply1”**. Seguidamente, en la línea **134** calculamos y guardamos en una variable (**posición**) en qué posición se encuentra la palabra **“VALOR=10”** con la función **“indexOf”**, la línea de código quedaría de la siguiente manera:

“**posición=controlReply1.indexOf("VALOR=")**”

Ya con estos datos se pude truncar o verificar el valor arrojado por el servidor. Luego, en la línea **135** se le suma **seis (6)** a la variable **“posición”**, ya que, solo se quiere tomar los valores que estando luego de la palabra **“VALOR=”**, es decir, en la variable **“posición”** se almacena la posición inicial donde está **“VALOR=”**.

Ejemplo:

Si la **posición** de la palabra “**VALOR=**” es igual a **36** y se desea leer solo los valores arrojados que están luego de la palabra “**VALOR=10**”, se le debe sumar **seis (6)** a la variable **posición** para luego así obtener la posición de los valores introducido por el usuario para cada **LED**. En la siguiente imagen se puede apreciar lo indicado en el ejemplo.



Ahora bien, ya se tiene a partir de qué posición están los valores de los **LED** introducidos por el usuario, con esto se pude saber en qué posición se encuentra el valor del **LED1** y el valor del **LED2** tal como se puede observar en línea **136** y **137** del código.

```

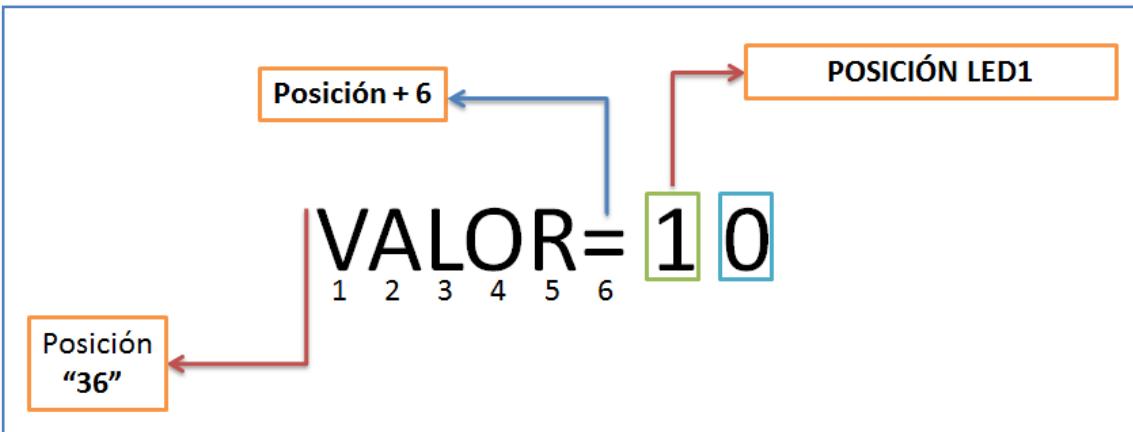
136.     posicionLED1=posicion+1; // se obtiene la posicion del
        valor del LED1
137.     posicionLED2=posicion+2; // Se obtiene la posicion del
        valor del LED2
    
```

En la línea 138 del código se aprecia la condición IF para tomar la decisión de encender el **LED1**, esta decisión se realiza comparando la variable “**posición**” y la variable **posiciónLED1**, con la función “**substring**” la función seria aplicada de la siguiente manera.

```

138.     if (controlReply1.substring(posicion, posicionLED1) == "1") // 
    a partir de posicion+6 en adelante hasta posicion LED1.
139.     {
140.         Serial.println("LED_1 Encendido");
141.         digitalWrite(LED1, HIGH);
142.     }else if(controlReply1.substring(posicion, posicionLED1)
143.             == "0")
144.             {
145.                 Serial.print("LED_1 Apagado");
146.                 digitalWrite(LED1, LOW);
}

```

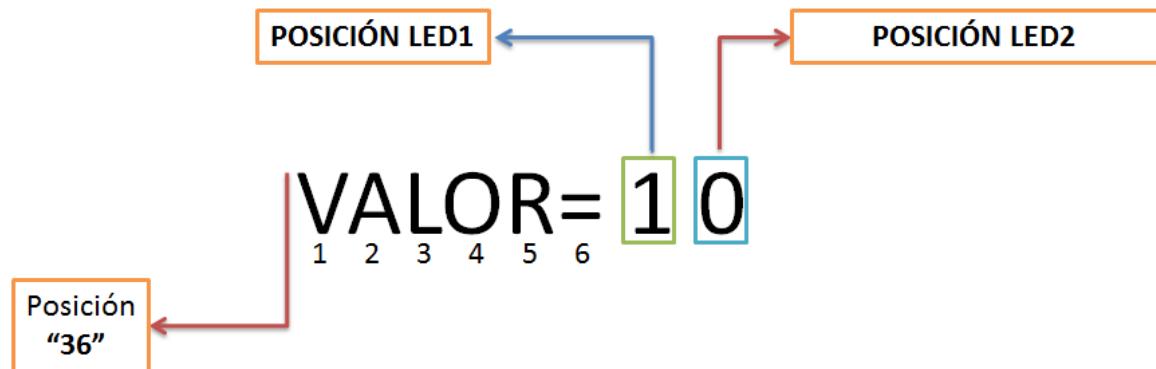


De esta manera se pude truncar las posiciones para ser comparada y tomar decisiones dentro del software, es decir, se compara si existe un “1” o un “0” entre las variables “**posición**” y “**posiciónLED1**”, tal como se muestra en el código anterior. Para el **LED2** se realiza el mismo procedimiento.

```

161.     if (controlReply1.substring(posicionLED1, posicionLED2) == "1")
    // A partir de posicion LED1 hasta posicion LED2
162.     {
163.         Serial.println("LED_2 Encendido");
164.         digitalWrite(LED2, HIGH);
165.     }else if(controlReply1.substring(posicionLED1, posicionL
    ED2) == "0")
166.     {
167.         Serial.println("LED_2 Apagado");
168.         digitalWrite(LED2, LOW);
169.     }
170.
171. }

```



RUTINA ENVIAR DATOS A MYSQL Y ENCENDER LED MEDIANTE WEB

Para la siguiente rutina solo se insertó como función, la rutina [recibir LED](#) en la rutina de enviar datos a **Mysql**. La función [recibir LED](#) se ejecuta cuando la bandera **i** es igual a tres (**i=3**), es decir, luego de enviar datos analógico a la base de datos **Mysql**.

```
1. // En esta version se usa el puerto serial 1 para comunicarse
   con el sim900
2. ///////////////////////////////////////////////////////////////////
3. int
   i=0,mostrar=0,e=0,x=0,segundos=0,segundosSD=0,ia=0,conexion=0,Er
   rorReply=0,NumReinicios=0;
4. float k=0;
5. int addr=0, Reinicios=0,DatosEnSD=0;
6. String control1,control,control2="STATE: CONNECT OK",Clear,show;
   // Se declara la variable control2 para comparar la conexió
7. String controlReply="";
8. String controlReply1="";
9. int Reply=0;
10.    // Librerias SD Y RTC/////////////////////////////
11.    #include <SPI.h>
12.    #include <SD.h>
13.    #include <Wire.h> // Iniciar I2C
14.    #include "RTClib.h" // Libreria para el RTC
15.    RTC_DS1307 RTC;
16.    const int chipSelect = (10,11,12,13); //Definimos el
   chipSelect, como es el Arduino Mega2560 se recomienda definirlo
   de la siguiente manera
17.    File dataFile;
18.    /////////////////////////////////
19. void setup()
20. {
21. i=0;
22. RTC.begin(); //Iniciando RTC
23. Wire.begin(); // Iniciando I2C
24. Serial.begin(19200);
25. Serial1.begin(19200);
26. // onGPRS();
27. delay(10000); //tiempo para ingresar a la red telefonica
28. Serial.println("Serial ON\r\n");
29. delay (100);
30. ////////////////////CONFIGURANDO
   INTERRUPCIONES/////////////////////////////
31. cli(); // Desabilitar interrupciones globales
32. TCCR4A=0; //Iniciando Registro
33. TCCR4B=0; //Iniciando Registro
34. OCR4A=15624; // Valor el cual timer se va a comparar con el
   CTC para reiniciar en 1 Seg
35. TCCR4B |=(1<<WGM42); // Indicar la comparación con el CTC
36. TCCR4B |=(1<<CS40); //Activar el prescaler a 1024 conjunto
   con el CS42
37. TCCR4B |=(1<<CS42); //Activar el prescaler a 1024 conjunto
   con el CS40
38. sei(); // Activar las interrupciones
39. //////////////////Configurando PIN Y
   SD///////////////////////////////
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
40.     pinMode(53, OUTPUT); // Para el SS o CS de la SD
41.     pinMode(4, OUTPUT); // Para LED1
42.     pinMode(3, OUTPUT); // Para LED2
43.     verificarsd();
44.     //////////////////Configurando RTC////////////////////////////
45.     if (!RTC.isrunning())
46.     {
47.         Serial.println("RTC ERROR");
48.         // Le asigna hora y fecha al RTC con la ultima compilacion
        del SKETCH se recomienda volcar el programa de una vez al
        arduino
49.         // Es decir se carga una sola vez y luego se comenta
50.         //RTC.adjust(DateTime(__DATE__, __TIME__));
51.     }
52. }
53. ///////////////////////////////////////////////
54. void loop()
55. {
56. inicio:
57.     Serial.flush();
58.     Serial1.flush();
59.     if(i==0)
60.     {
61.         conectgprs(); // Funcion conectar GPRS se ejecuta solo
        cuando i=0
62.     }
63.     if (i==1)
64.     {
65.         Read(); //Funcion Read se ejecuta cuando i=1
66.     }
67.     if(i==2)
68.     {
69.         segundos=0;
70.         TIMSK4 =(1<<OCIE4A); // activando interrupcion por
        comparacion
71.         i=3;
72.     }
73.     if (i==3)
74.     {
75.         if(i!=0&&i!=1) // Condiciona que no entre a esta opcion,
        cuando i=0 o i=1, ya que, el programa es secuencial. Es decir,
        si i=0 despues de ejecutar la funcion, de igual manera ejecutara
        la siguiente funcion recibirLED
76.     }
77.     enviardatoAnalog(0); // Llama a la funcion enviar dato
        analogico, AIO
78. }
79. if(i!=0&&i!=1)
80. {
81.     recibirLED(4,3); // Llama la funcion para encender los LED
82. }
83. }
84. if(i==20)
85. {
86.     reiniciarGPRS();
87. }
88. if(i==30)
89. {
90.     if(DatosEnSD==1)
```

```
91.      {
92.        k=LeerAnalog(0);
93.        sdw("Temp.txt"," Temp: ",k);
94.        DatosEnSD=0;
95.      }
96.    }
97.  }
98. ////////////////FUNCION CONECTAR A
GPRS///////////////
99. void conectgprs()
100. {
101.   if(Serial.available()==0) // Verificar Serial disponible
102.   {
103.     TIMS4 =(0<<OCIE4A); // Desactivar interrupcion
104.     Serial.println("Conectando..");
105.     delay(300);
106.     Serial1.print("AT+CGATT?\r\n"); // Comando AT para entrar a
conexion TCP
107.     clearBuffer();
108.     delay(1000);
109.     Serial1.print("AT+CSTT=\\"web.tmovil.cl\\",\\"web\\",\\"web\\\"\r\
n"); // Configurar APN, USUARIO y CONTRASEC
110.     clearBuffer();
111.     delay(1000);
112.     Serial1.print("AT+CIICR\r\n"); // Levantar conexion GPRS
113.     clearBuffer();
114.     delay(1000);
115.     Serial1.print("AT+CIFSR\r\n"); // Obtener direccion IP
116.     clearBuffer();
117.     delay(1000);
118.     Serial1.print("AT+CIPSTART=\\"TCP\\",\\"190.215.92.114\\",80\r\
n"); // Conectarse al servidor mediante TCP
119.     clearBuffer();
120.     delay(300);
121.     clearBuffer();
122.     i=1;
123.   }
124. }
125. /////////////////
126. ////////////LIMPIAR
BUFFER///////////
127. void clearBuffer()
128. {
129.   while (Serial1.available() !=0)
130.   {
131.     Serial.println(Serial1.readString()); //Usado para cuando
se quiere leer a respuesta del sim900, para ello, se tiene que
comentar las 3 lineas siguientes y descomentar esta linea.
132.     Clear=Serial1.readStringUntil('\n'); // Limpando Buffer,
Guardando respuesta en la variable Clear
133.     Clear=""; // Iniciar variable en 0
134.   }
135. }
136. /////////////////
137. ////////////VERIFICACION DE
CONEXION///////////
138. void Read()
139. {
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
140.    Serial.println("Verificando Conexion");
141.    if(Serial1.available()==0)
142.    {
143.        TIMSK4 = (0<<OCIE4A); // Desactivar interrupcion
144.        Serial1.println("AT+CIPSTATUS\r\n");// Envia el comando
145.        delay(100);
146.        while (Serial1.available() !=0)// Mientras el buffer este
147.            lleno de datos se queda en while
147.        {
148.            control=Serial1.readStringUntil('\n');// Lee la respuesta
149.            del modulo SIM900
150.            delay(5);
150.            control1=control; // Pasamos la variable control a control1
151.            para luego compararla
151.            control.trim();// Corta los espacio del string
152.            control1.trim();
153.            Serial.println(control);
154.        }
155.        if(control1.equals(control2))// Compara la respuesta con
155.        control2 = "STATE: CONNECT OK"
156.        {
157.            Serial.println("Conexion Exitosa");
158.            i=2;
159.            conexion=0;
160.        } else
161.        {
162.            Serial.println("Conexion Fallida");
163.            i=0;
164.            conexion++;
165.        }
166.    }
167.    if (conexion==5)
168.    {
169.        i=20;
170.    }
171.    clearBuffer();
172.    control=""; // Iniciar la variable en 0 para la proxima
172.    comparacion
173.    control1="";
174.    }
175.    /////////////////////////////////
176.    //////////////////Funcion Interrumpir para verificar
176.    Status/////////////////
177.    ISR(TIMER4_COMPA_vect) // ISR TIMER4_COMPA_vect activa la
177.    funcion cuando la comparacion del CTC y el timer es 15624
178.    {
179.        segundos++;
180.        segundosSD++;
181.        if(segundos==120)// Entra a la condicion luego de 2
181.        minutos.
182.        {
183.            i=1;
184.            segundos=0;
185.        }
186.        if(segundosSD==10)
187.        {
188.            DatosEnSD=1;
189.            segundosSD=0;
```

```
190. }
191. }
192. ///////////////////////////////////////////////////
193. ///////////////////////////////////////////////////Enviar Datos
    Analogicos/////////////////////////////////////////////////
194. void enviardatoAnalog(int y)
195. {
196. if(Serial1.available()==0)
197. {
198. x=analogRead(y); //Se lee el puerto analogico a traves por
    paso de parametro de la funcion en este caso el puerto analogico
    0
199. Serial1.println("AT+CIPSEND\r\n"); // Preparar el mensaje a
    enviar
200. delay(1000);
201. Serial1.print("GET /arduino/registros.php?temperatura=");
    // Escribir mensaje a enviar al servidor mediante el metodo GET,
    se envia sin fin de linea para poder enviar el dato "x", luego
    de enviar el dato se envia fin de linea y retorno de carro "\r\n"
202. delay(300);
203. Serial1.print(x);
204. delay(300);
205. Serial1.print(" HTTP/1.1\r\n"); // Se envia la version del
    protocolo, para este caso HTTP/1.1
206. delay(1000);
207. Serial1.print("Host: 190.215.92.114\r\n"); // De acuerdo al
    protocolo HTTP/1.1 en la cabecera debe especificar el host
    receptor
208. delay(1000);
209. Serial1.print("Connection: Keep-Alive"); // El protocolo
    HTTP/1.1 permite conexiones persistente, es decir, la conexion
    queda abierta durante el tiempo programado en el servidor. Pues,
    ya que el servidor responde a la peticion y cierra la
    conexion.Sin embargo, esta configuracion se realizo en el
    servidor y se puede omitir esta linea de codigo
210. delay(1000);
211. Serial1.println("\r\n"); // Entre linea
212. Serial1.println("\xA"); // Caracter especial para enviar
    mensaje al servidor
213. delay(50);
214. WaitReplyError(); // Verifica si hay error en el envio del
    dato
215. }
216. }
217. ///////////////////////////////////////////////////
218. /////////////////////////////////////////////////// APAGAR GPRS EN CASO DE FALLAR
    CONEXION/////////////////////////////////////////////////
219. void offGPRS()
220. {
221. delay(3000);
222. clearBuffer();
223. if(Serial1.available()==0)
224. {
225. Serial1.println("AT+CPOWD=1");
226. delay(100);
227. clearBuffer();
228. }
229. }
```

```
230.     void onGPRS()
231.     {
232.         Serial.println("Enciendo SIM900...");
233.         delay(3000);
234.         pinMode (9,OUTPUT);
235.         digitalWrite (9, LOW);
236.         delay(1000);
237.         digitalWrite (9, HIGH);
238.         delay(2000);
239.         digitalWrite(9,LOW);
240.         delay(3000);
241.         conexion=0;
242.         delay(3000);
243.         clearBuffer();
244.     }
245.     /////////////////////////////////
246.     /////////////////////FUNCION LEER RESPUESTA///////////////////
247.     void WaitReplyError()
248.     {
249.         controlReply="";
250.         controlReply1="";
251.         while (Serial1.available()!=0)// Mientras el buffer este
252.             lleno de datos se queda en while
252.         {
253.             controlReply=Serial1.readStringUntil('\r');// Lee la
254.             respuesta del modulo SIM900
254.             delay(5);
255.             controlReply.trim();
256.             controlReply1=controlReply;
257.             controlReply1.trim();
258.             Serial.println(controlReply1);
259.             if(controlReply1.equals("ERROR") || controlReply1.equals("SEN
260.             D FAIL"))// Compara la respuesta con controlReply
260.             {
261.                 Serial.println("Envio Faillido");
262.                 ErrorReply++; // Si hay un error en el envio se incrementa
263.                 ErrorReply
263.                 if(ErrorReply==2) // Si hay dos errores en el envio se
264.                     coloca la bandera i=1, para chequear la conexion con el servidor
264.                     {
265.                         ErrorReply=0;
266.                         i=1;
267.                         goto n;
268.                     }
269.                 }
270.                 if(controlReply1.equals("Connection: close"))// Compara la
271.                     respuesta con controlReply
271.                     {
272.                         Serial.println("Cerrando Conexion");
273.                         i=0;
274.                     }
275.                 }
276.                 n:
277.                 Serial1.flush();
278.                 return 0;
279.             }
280.             //////////////////Funcion Grabar en
281.             SD///////////////////////
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
281.     void sdw(char filename[], String z, int x) // PASO EL
        PARAMETRO DEL FILE
282.     {
283.         DateTime now = RTC.now();
284.         dataFile=SD.open(filename,FILE_WRITE);
285.         Serial.print("Registrando en ");
286.         Serial.write(filename);
287.         Serial.println('\n');
288.         dataFile.print(z); // Referencie de lo que se vaa a guardar
289.         dataFile.print(x); // variable que se va a guardar
290.         dataFile.print(' ');
291.         dataFile.print(now.year(), DEC);
292.         dataFile.print('/');
293.         dataFile.print(now.month(), DEC);
294.         dataFile.print('/');
295.         dataFile.print(now.day(), DEC);
296.         dataFile.print(' ');
297.         dataFile.print(now.hour(), DEC);
298.         dataFile.print(':');
299.         dataFile.print(now.minute(), DEC);
300.         dataFile.print(':');
301.         dataFile.println(now.second(), DEC);
302.         dataFile.flush();
303.         dataFile.close();
304.     }
305.     /////////////////////////////////
306.     /////////////////////Funcion Verificar
307.     SD/////////////////////////////
307.     void verificarsd()
308.     {
309.         if(!SD.begin(chipSelect))
310.         {
311.             Serial.println("Fallo en tarjeta SD, Verificar Cableado o
            Tarjeta SD");
312.         }else
313.         {
314.             Serial.println("Tarjeta SD OK");
315.         }
316.     }
317.     /////////////////////////////////
318.     /////////////////////Leer Dato
319.     Analogico///////////////////
319.     float LeerAnalog(int y)
320.     {
321.         float x;
322.         x=analogRead(y);
323.         return x;
324.     }
325.     /////////////////////////////////
326.     /////////////////////LEER SALIDAS
327.     DIGITALES///////////////////
327.     void recibirLED(int LED1, int LED2)
328.     {
329.         int posicion=0, longitud=0, posicionLED1=0, posicionLED2=0;
330.         if(Serial1.available() == 0)
331.         {
```

BITÁCORA SOBRE PROYECTO GPRS (ARDUINO MEGA2560 Y SIM900)

```
332.     Serial1.println("AT+CIPSEND\r\n"); // Preparar el mensaje a
enviar
333.     delay(500);
334.     Serial1.print("GET /LED/GetStateLED2"); // Escribir mensaje
a enviar al servidor mediante el metodo GET, se envia sin fin de
linea para poder enviar el dato "x", luego de enviar el dato se
envia fin de linea y retorno de carro "\r\n"
335.     delay(100);
336.     Serial1.print(" HTTP/1.1\r\n"); // Se envia la version del
protocolo, para este caso HTTP/1.1
337.     delay(100);
338.     Serial1.print("Host: 190.215.92.114\r\n"); // De acuerdo al
protocolo HTTP/1.1 en la cabecera debe especificar el host
receptor
339.     delay(100);
340.     Serial1.print("Connection: Keep-Alive"); // El protocolo
HTTP/1.1 permite conexiones persistente, es decir, la conexion
queda abierta durante el tiempo programado en el servidor. Pues,
ya que el servidor responde a la peticion y cierra la
conexion. Sin embargo, esta configuracion se realizo en el
servidor y se puede omitir esta linea de codigo
341.     delay(500);
342.     Serial1.println("\r\n"); // Entre linea
343.     Serial1.println("\xA");
344.     WaitReplyError(); // Verifica si hay error en el envio del
dato
345.     longitud=controlReply1.length(); // Se obtiene la longitud
del texto enviado por el servidor
346.     posicion=controlReply1.indexOf("VALOR="); // Se obtiene la
posicion de VALOR=
347.     posicion=posicion+6; // Se le suma 6, ya que "VALOR=" tiene
6 caracteres y solo queremos leer los valores arrojados.
348.     posicionLED1=posicion+1; // se obtiene la posicion del
valor del LED1
349.     posicionLED2=posicion+2; // Se obtiene la posicion del
valor del LED2
350.     if (controlReply1.substring(posicion,posicionLED1)=="1")// 
// A partir de posicion+6 en adelante hasta posicion LED1.
351.     {
352.         Serial.println("LED_1 Encendido");
353.         digitalWrite(LED1,HIGH);
354.     }else
355.     {
356.         Serial.println("LED_1 Apagado");
357.         digitalWrite(LED1,LOW);
358.     }
359.     if
360.     {
361.         Serial.println("LED_2 Encendido");
362.         digitalWrite(LED2,HIGH);
363.     }else
364.     {
365.         Serial.println("LED_2 Apagado");
366.         digitalWrite(LED2,LOW);
367.     }
368. }
```

```
369.     controlReply="";
370.     controlReply1="";
371.   }
372.   /////////////////////////////////
373.   ///////////////////Reiniciar
374.   void reiniciarGPRS()
375.   {
376.     Serial.println("Reiniciando SIM900...");
377.     offGPRS();
378.     delay(3000);
379.     onGPRS();
380.     Reinicios++; // Bandera para comparar en la condicion i=30
381.     NumReinicios++; // Para indicar en la SD cuantas veces se
382.     ha reiniciado
383.     sdw("Reset.txt"," Sim900 Reiniciado #",NumReinicios); // Se
384.     // pasa el parametro "nombre del archivo","Lo que se desea
385.     // escribir"," Variable que se quiera reflejar"
386.     if(Reinicios==2) // Verificar el numero de reinicios para
387.     // asi grabar en la SD
388.     {
389.       i=30; // colocar la bandera 30, esto indica dentro el loop
390.       // principal que se iniciar la funcion guardar en tarjeta sd.
391.       Reinicios=0;
392.       Serial.println("Registrando datos en tarjeta SD"); //
393.       // Registrando datos en la tarjeta SD
394.       TIMSK4 = (1<<OCIE4A); // activando interrupcion por
395.       // comparacion, para verificar luego de 2 minutos si hay conexion
396.       // con el servidor.
397.     }else
398.     i=0;
399.   }
```

Cabe destacar que, en el código cuando se llama la función para enviar dato analógico y para recibir LED se condicionaron mediante **if**, ya que, el software es secuencial. Es decir, si la bandera **i** es igual a uno (**i=1**) después de ejecutar la función **enviar datos analógico**, no se debería ejecutar la función **recibir LED**, pues, cuando la bandera **i** es igual uno (**i=1**), significa que se debe chequear la conexión con el servidor, ya que, hubo un error de envíos de datos al mismo. Por ende, no se debe enviar datos hasta chequear la conexión con el servidor. Es por ello, que se condicionan dichas funciones.