Electrónica General y Aplicada

Guía de Comunicación Serie para el Trabajo Integrador

Introducción

En el proyecto integrador se solicita a los grupos que realicen un automatismo basado en microcontrolador (μC). Éste puede ser por ejemplo el control de nivel de un tanque, de un motor, de un ascensor, una alarma etc.
Para esto se debe trazar el esquema de conexiones del μC con el sistema a controlar, hacer un **programa** en un lenguaje de alto nivel (normalmente C), que se compilará (traducción al lenguaje del μC) y se grabará en el μC.
Este **programa** debe comprobar las **entradas** del μC (analógicas o digitales) y activar/desactivar **salidas** de acuerdo con un algoritmo y/o una lógica de funcionamiento.
Por ejemplo para automatizar un ascensor el programa debe comprobar las entradas conectadas a pulsadores y detectores de posición, y activar/desactivar las salidas conectadas a los transistores o relés que comandan la marcha del motor en ambos sentidos (para ascenso o descenso de la cabina).
Esto se puede esquematizar en forma genérica como muestra la figura 1, y consiste en un **automatismo no supervisado**.

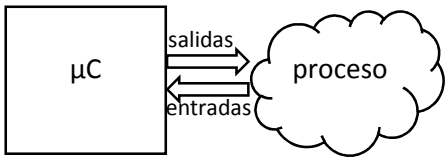


Figura 1. Automatismo no supervisado

En la industria y en muchos otros ámbitos se procura que los automatismos puedan ser supervisados desde un sistema central, como una aplicación en una PC (ejemplo un software SCADA, un supervisor de sistema mecatrónico). Esto se estudiará en las unidades 8 y 9.

Un **automatismo supervisado** permite que al controlador se le pueda solicitar información, ajustar parámetros e incluso dar órdenes directas (operación remota). En el automatismo del ascensor, el supervisor podría, por ejemplo:

1. preguntar al μC la cantidad de veces que se accionó el motor, para estadísticas de uso y/o para programar tareas de mantenimiento.
2. modificar el tiempo durante el cual las puertas deben permanecer abiertas.
3. bloquear un piso determinado (que el ascensor no se pueda detener en ese piso aunque el usuario lo solicite).
4. llevar el ascensor a un piso determinado (operación remota).
5. preguntar en qué piso está el ascensor
6. activar/desactivar el automatismo.

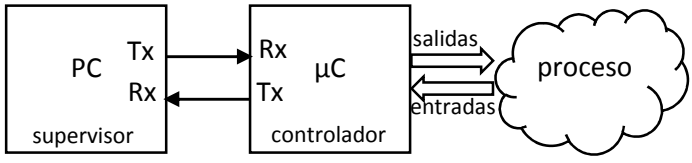


Figura 2. Automatismo supervisado

La tarea de supervisión implica entonces el intercambio de información entre el supervisor y el controlador.
Podemos reconocer en estos ejemplos que 1 y 5 son **solicitudes** de información del supervisor que deben ser respondidas por el controlador, 2 es un parámetro o **consigna** numérica (tiempo de apertura de puerta) que el supervisor pasa al controlador, 3 y 4 se pueden considerar también

consignas numéricas (piso a bloquear o piso de destino) que el supervisor debe pasar al controlador, mientras que 6 es un cambio de estado o consigna que no requiere parámetros.

Este intercambio se realiza a través de un **canal de comunicación** serie entre ambos equipos. Ambos equipos deben disponer entonces de un hardware de comunicación serie compatible, y deben acordar el formato de la información.
El formato de la información se denomina **protocolo de comunicación.** Como se estudiará en la unidad 8 existe una gran variedad de protocolos industriales, cada uno con características que lo hacen más o menos apropiado para distintas aplicaciones.

Desde el año 2009, con el objetivo de integrar la mayor parte de los contenidos de la asignatura en el Trabajo Especial, procuramos que los grupos deban realizar automatismos supervisados, es decir que incluyan la comunicación entre el µC y la PC.

Comunicación entre PC y microcontrolador.

Para implementar el **canal de comunicación** serie conectaremos un puerto serie de la PC (puerto COM) con un puerto serie del µC (interfaz UART). Las PC domésticas actuales – portátiles y de escritorio – no disponen de conectores de estos puertos, por lo que en ese caso utilizaremos un adaptador USB/serie (provisto por la cátedra), que crea un puerto COM virtual. Si fuera necesario se realizará además la adaptación de niveles eléctricos, como se explicará más adelante. Como se muestra en la figura 2, los mensajes de la PC al µC van del transmisor (Tx) de la PC al receptor (Rx) del µC, y los mensajes del µC a la PC van del Tx del µC al Rx de la PC.

Estructura de mensajes.

Como **protocolo de comunicación** emplearemos un formato de mensajes propio muy sencillo, con las funciones mínimas para pasar las consignas y solicitudes necesarias, y que sea fácil de interpretar por el µC.
El formato de los mensajes será similar para todos los proyectos. Luego dependiendo de la aplicación cambiarán algunos elementos.

Mensaje Solicitud (PC → µC)

Un mensaje tipo **solicitud** de información tendrá una estructura que comprende la siguiente secuencia de caracteres.

- Carácter que indica el **inicio** de mensaje. Se usará el carácter "**:**" (ASCII 58)
- Número de controlador destinatario. Se usará un dígito de "0" a "9" (ASCII de 48 a 57). Este número es necesario porque el supervisor podría estar supervisando a muchos controladores a través del mismo canal o **bus**, como se verá en la unidad 8. Cada controlador tendrá una "dirección" entre "1" y "9". La dirección "0" se reserva para los mensajes que el supervisor quiera enviar simultáneamente a todos los controladores (mensaje de difusión o *broadcasting*), por ejemplo para inicializarlos.
- Nombre del parámetro o valor solicitado. Se usará una letra, por ejemplo "P" para piso, "T" para tiempo etc. Si hubiera más de un parámetro similar, por ejemplo más de un tiempo, podrá haber un segundo carácter (letra o número), por ejemplo "TA", "T1", "T2" etc.
- Carácter de final. Se usará el carácter denominado "retorno de carro" o CR en inglés (ASCII 13), también representado como '\r' en lenguaje C. Es un carácter no imprimible, que en un teclado se produce cuando se toca la tecla "Enter" o "Intro".

Así, en el ejemplo del ascensor, para que el supervisor consulte al controlador **3** en qué piso se encuentra el ascensor, el mensaje podría ser la sucesión de caracteres:

	':'	'3'	'P'	'\r'
Que en ASCII (expresado en decimal) será	58	51	80	13
Y expresado en hexadecimal	3A	33	50	0D

Mensaje respuesta a solicitud (µC → PC)

En respuesta al mensaje visto anteriormente, el controlador deberá transmitir un mensaje similar, pero que incluya la información solicitada, que en el ejemplo anterior sería el valor numérico que corresponde al piso en el que se encuentra el ascensor. Así, si se encontrara en el piso 14, la respuesta sería:

':'	'3'	'P'	'1'	'4'	'\r'

Es decir, la primera parte idéntica ":3P" y luego los caracteres "1" y "4". O expresada en ASCII decimal

58	51	80	49	52	13

Mensaje Consigna numérica (PC → µC)

Un mensaje tipo **consigna numérica** será en la primera parte similar al mensaje **solicitud**, pero agregando el valor numérico de la consigna, quedando así:

- Carácter de inicio ":"
- Número de controlador destinatario.
- Parámetro o valor a modificar ("P", "T", "X" etc).
- Valor numérico del parámetro, expresado como **cadena decimal**.
- Carácter de final. Se usará el carácter retorno de carro '\r'.

Así, en el ejemplo del ascensor, para que el supervisor le ordene al controlador 3 que el tiempo de apertura de la puerta sea 3200 milisegundos, el mensaje podría ser la sucesión de caracteres (observar las comillas):

':'	'3'	'T'	'3'	'2'	'0'	'0'	'\r'

Que en ASCII (expresado en decimal) será

58	51	84	51	50	48	48	13

Y expresado en hexadecimal

3A	33	54	33	32	30	30	0D

Mensaje respuesta a Consigna numérica (µC → PC)

Consiste simplemente en un "eco" del mensaje de la PC, y se efectúa para verificar si el mensaje de la PC fue leído correctamente por el microcontrolador. En el ejemplo anterior sería

':'	'3'	'T'	'3'	'2'	'0'	'0'	'\r'

Hardware de comunicación en el µC

En nuestros trabajos utilizamos generalmente microcontroladores que incluyen UART. Por ejemplo, el ATmega328 (presente en los arduino UNO y Nano), el pin 2 (PD0) es el pin de recepción serie RXD, y el pin 3 (PD1) es el de transmisión serie TXD.

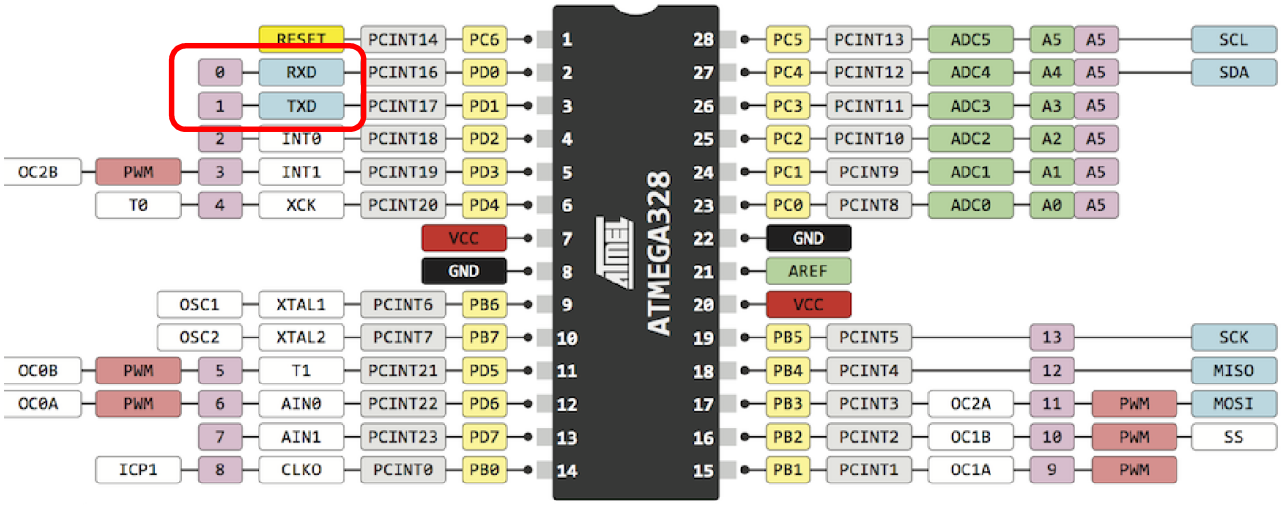


Figura 3. Pines del ATmega328P en encapsulado DIP 28. En amarillo claro los pines como GPIO, en violeta los números de pines en placa Arduino, el resto son funciones alternativas de los pines. En recuadro rojo los pines que sirven de Tx y Rx

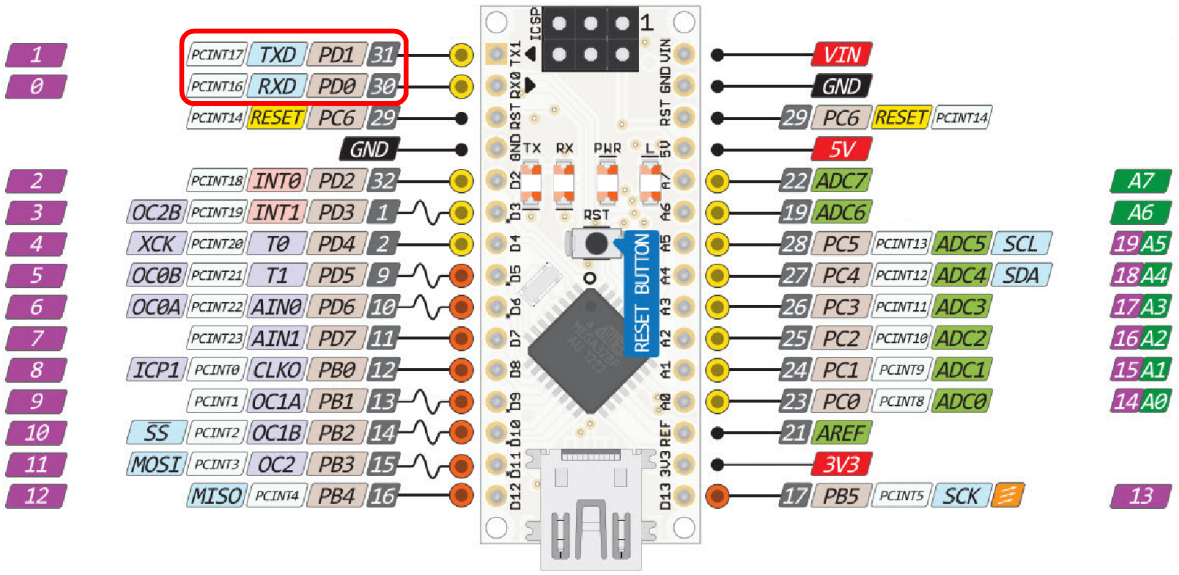


Figura 4. Pines del Arduino NANO. En en violeta los números de pines en placa Arduino, en gris la función básica de los pines (como Entrada o Salida), en celeste y azul claro otras funciones alternativas. En recuadro rojo los pines que sirven de Tx y Rx. En esta placa están conectados al puente USB/Serie (chip que está en la cara inferior de la placa).

Software de comunicación en el µC

Para habilitar estos pines como RXD y TXD, es decir para que el µC pueda comunicarse con un dispositivo serie externo, el programa en el µC debe **inicializar** el periférico UART escribiendo los valores apropiados en ciertos registros de control del microcontrolador. En el entorno Arduino la inicialización se realiza de manera muy simple mediante una instrucción como:

```
Serial.begin(9600);
```

Esta comunicación será a 9600 bps, 8 bits de datos y sin paridad. Se pueden utilizar otras velocidades, en vez de 9600, que tendrá que ser la misma del equipo con el cual se comunicará (PC, otro µC etc).

Al escribir la instrucción anterior, PD0 y PD1 pasan a ser los pines Rx y Tx respectivamente. En el ATmega328 alimentado a 5 volts, estos pines utilizan 5 volts para representar el '1' y 0 volts para el '0', lo que usualmente se denomina "**niveles TTL**".

Hardware de comunicación en PC

En los modelos antiguos de PC es posible disponer del conector de puerto serie RS232, como se muestra en la figura 5. En este conector el pin 2 es Rx, el pin 3 es Tx y el pin 5 es masa (los números están en relieve en el plástico interior).

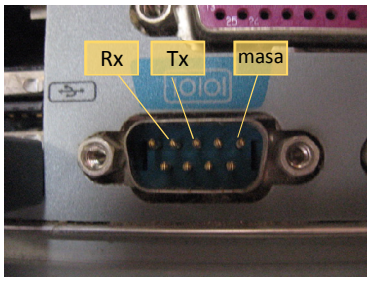


Figura 5. Puerto COM (9 pines) en una PC de escritorio

Como los voltajes de las señales RS232 están invertidos respecto a las del microcontrolador (el "0" se representa con +12 volts y el "1" con –12 volts), se requiere un adaptador eléctrico como el integrado MAX232 (figura 6).

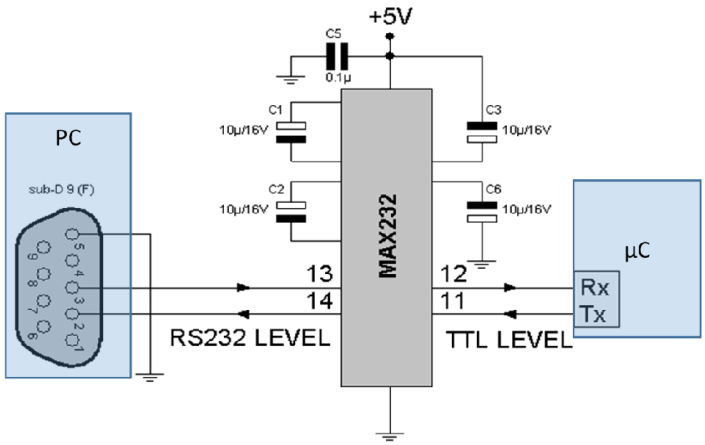


Figura 6. Adaptador MAX232 para conexión entre un µC y una PC que tiene puerto serie RS232

En las PCs de escritorio y notebooks actuales de uso doméstico no está disponible el puerto RS232. En tal caso se puede utilizar un adaptador USB/Serie como los mostrados en la figura 7. Los adaptadores pueden ser de USB a norma RS232 (izquierda) o a TTL (derecha). En el primer caso se debe conectar también un adaptador eléctrico como los vistos en 5a o 5b, en el segundo se puede conectar directamente al microcontrolador.

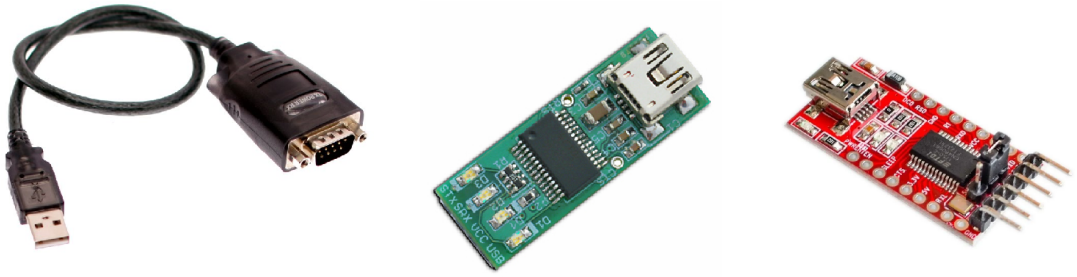


Figura 7. (a) Adaptador USB/Serie (RS232) (b) Adaptadores USB/Serie

Las placas Arduino (Por ejemplo UNO o Nano) ya incorporan el chip adaptador USB/Serie TTL, conectado a los pines RXD y TXD.

Software de comunicación en PC. Terminal

Así como en el microcontrolador debe abrirse el puerto de comunicaciones inicializando la UART, en la PC debe abrirse el puerto COM que se va a utilizar para enviar y recibir los mensajes. Para abrir el puerto, enviar y recibir mensajes podemos utilizar la herramientas del entorno Arduino denominada "Monitor Serie", que es un software de terminal básico. Hay otras aplicaciones de terminal más completas. También hay funciones de comunicación en aplicaciones como Matlab, Labview, Qt, o se pueden programar en aplicaciones propias.

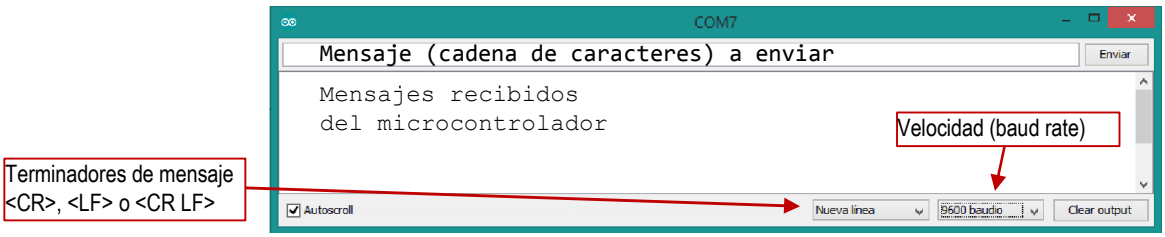


Figura 8. Herramienta *Monitor serie* del entorno Arduino

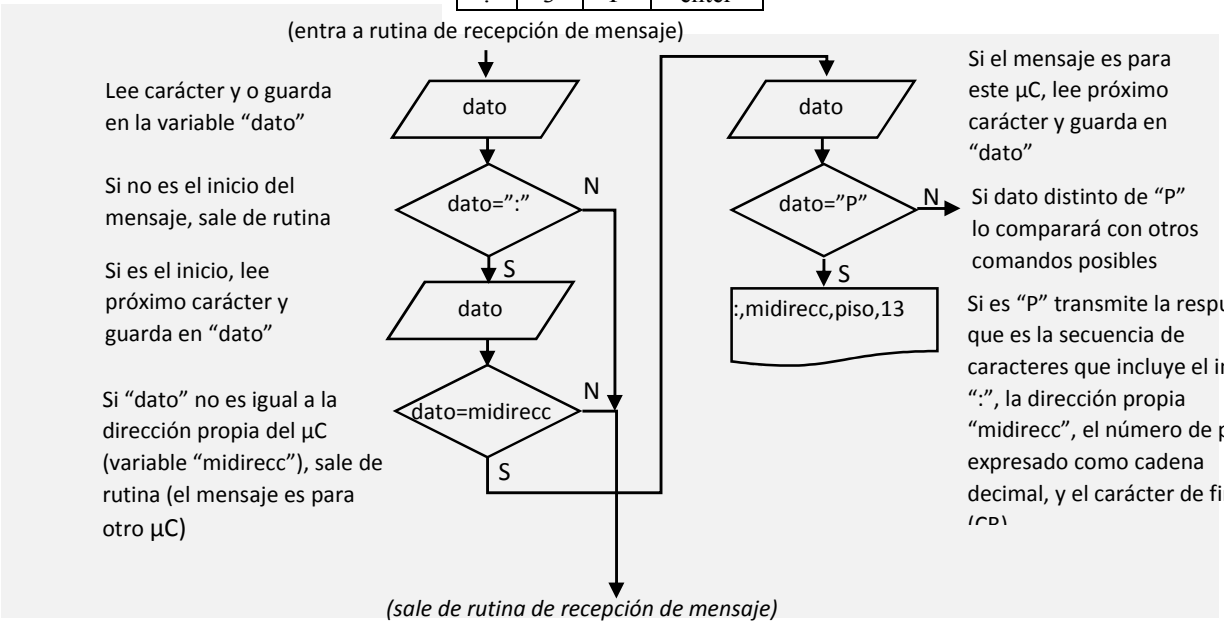
Intercambio de mensajes entre PC y microcontrolador

Los mensajes que se explicaron anteriormente (solicitudes, consignas) son secuencias de caracteres que deben ser interpretadas por el programa que corre en el µC. Hay dos maneras de interpretar un mensaje:

- **Sin *buffer*:** Una vez que se recibió el carácter de inicio ":", se va procesando cada carácter que se va recibiendo para identificar comando, parámetros etc.
- **Con *buffer*:** Una vez que se recibió el carácter de inicio ":", se van guardando en posiciones consecutivas de memoria RAM (vector, *array* o *buffer*) los caracteres que se van recibiendo hasta el carácter de final (ASCII 13). Recién entonces se procede a procesar cada carácter del mensaje (cada byte copiado en RAM) para identificar comando, parámetros etc.

Explicaremos el primer método para interpretar un mensaje (sin *buffer*).
Supongamos un mensaje como la solicitud vista anteriormente:

":"	"3"	"P"	<enter>



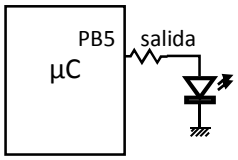
En resumen:
Para desarrollar un automatismo supervisado se requiere que el controlador pueda comunicarse con un supervisor, usualmente una PC, aunque también puede ser un PLC, terminal etc.
Para crear el canal de comunicación debe configurarse hardware y software, tanto en la PC como en el microcontrolador, compatibilizando niveles eléctricos y acordando la velocidad de transmisión, paridad etc.
Para intercambiar información mediante mensajes (solicitudes, consignas y respuestas) debe establecerse un formato o protocolo de comunicación.
En el microcontrolador hay que programar las rutinas capaces de interpretar estos mensajes, procesando cada carácter. Este procesamiento puede ser realizado a medida que van llegando (opción básica pero no recomendable) o una vez que se ha recibido todo el mensaje en un *buffer*.

Ejemplos

Para afirmar los conceptos vistos vamos a realizar ejemplos progresivos de programas en microcontrolador. En todos los casos el circuito es el mismo. Se resolverán todos los ejemplos en el entorno Arduino .

1. Generador de pulsos simple de período fijo 1Hz
   Es el conocido *blink*, el "Hola mundo" de los microcontroladores. El propósito es ver las partes esenciales de un programa para comandar un pin de salida.
2. Generador de pulsos de período fijo con cambio de modo.
   Al ejemplo anterior se agrega la lectura de señales binarias externas para arranque y parada del oscilador.
3. Generador de pulsos "supervisado por **puerto serie**", con comunicación simple.
   Se agrega la posibilidad de solicitar información, pasar parámetros y operar en forma remota, desde un supervisor.
4. Generador de pulsos "supervisado por **puerto serie**", con comunicación asistida por la función serialEvent.
5. Generador de pulsos "supervisado por **puerto serie**", con comunicación con *buffer* e interrupción.

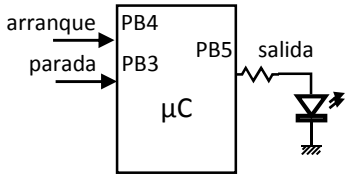
Ejemplo 1: Oscilador simple de período fijo 1 Hz



Se dispone de una salida en PB5, que en las placas Arduino modelos Nano y UNO corresponde a la salida 13, que tiene un LED de testigo.
El programa simplemente configura dicho pin como salida, y luego lo enciende y apaga cíclicamente con un retardo de 500 ms entre ambas acciones

```
#define salida 13              // Es PB5 y Led de placa UNO o Nano

void setup()                   // Función de configuración de Arduino
{
   t=500;                      // la variable t es el semiperíodo
   pinMode(salida, OUTPUT);
}

void loop()                    // Función cíclica de control de Arduino
{
     digitalWrite(salida, HIGH);
     delay (t);
     digitalWrite(salida, LOW);
     delay(t);
}
```

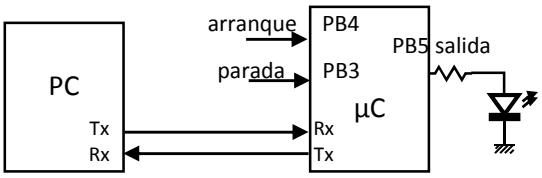
Ejemplo 2: Oscilador de período fijo 1 Hz, con cambio de modo mediante pulsadores



Se dispone de dos entradas (arranque y parada) y una salida (pulsos). Al comenzar el programa el generador está en estado *ciclo*, y comienza a producir pulsos de período 1000 ms. Si se pulsa la entrada "parada" el oscilador pasa al estado *reposo*, y deja de entregar pulsos. Si se pulsa la entrada "arranque", el generador vuelve al estado *ciclo*.

```
#define arranque 11            // Es PB3 en UNO o Nano (ver pines en figura 3
#define parada 12              // Es PB4
#define salida 13              // Es PB5 y Led de placa UNO o Nano

enum tEstado{ciclo, reposo}; // Variable enumerada para representar Estado ciclo o reposo
enum tEstado Estado=ciclo;

int t;

void setup()                   // Función de configuración de Arduino
{
   t=500;                      // la variable t es el semiperíodo
   pinMode(arranque, INPUT);
   pinMode(parada, INPUT);
   pinMode(salida, OUTPUT);
}

void loop()                    // Función cíclica de control de Arduino
{
switch (Estado)
  {
  case ciclo:                                          // En estado activo genera pulso…
     digitalWrite(salida, HIGH);
     delay (t);
     digitalWrite(salida, LOW);
     delay(t);
     if (digitalRead(parada) == HIGH) Estado = reposo;  // … y verifica pulsador de parada
     break;
  case reposo:                                  // En reposo solo verifica pulsador de arranque
     if (digitalRead(arranque) == HIGH) Estado = ciclo;
     break;
  default:
     break;
  }
}
```

Ejemplo 3: Oscilador "supervisado por **puerto serie**"



Este es un ejemplo básico de automatismo supervisado. Dispone de arranque y parada como el ejemplo 2, pero ahora través del puerto serie se puede supervisar el funcionamiento:
Este ejemplo básico interpreta mensajes simples, sin delimitador de inicio ":" ni número de controlador.
Con el mensaje "C\r" se **solicita** la cantidad de pulsos generados. Debe responder con un mensaje "Cnnn\r", por ejemplo "C235\r".
Con el mensaje "Tnnnn\r se envía la **consigna** cantidad de milisegundos del semiperíodo. Por ejemplo T100\r configura el oscilador 200 milisegundos de período (5Hz). Debe responder replicando la consigna, para que el supervisor verifique que fue comprendida.

```
#define arranque 11           // Es PB3 en UNO o Nano (ver pines en figura 3
#define parada 12             // Es PB4
#define salida 13             // Es PB5 y Led de placa UNO o Nano

enum tEstado{ciclo, reposo}; // Variable enumerada para representar Estado ciclo o reposo
enum tEstado Estado=ciclo;
int contador=0;               // contador de pulsos
int t;                        // tiempo de semiperíodo
int total;                    // variable para convertir cadena decimal en nñumero binario

void setup()                  // Función de configuración de Arduino
{
   t=500;                     // inicializa en 500 ms (período 1 segundo)
   Serial.begin(9600);        // Inicia objeto Serial (para usar sus funciones)
   pinMode(arranque, INPUT);
   pinMode(parada, INPUT);
   pinMode(salida, OUTPUT);
}

void loop()                   // Función cíclica de control de Arduino
{
switch (Estado){
  case ciclo:                                 // En estado activo genera pulso…
     contador++;
     digitalWrite(salida, HIGH);
     delay (t);
     digitalWrite(salida, LOW);
     delay(t);
     if (digitalRead(parada) == HIGH) Estado = reposo;      // … y verifica pulsador de parada
     break;
  case reposo:                                // En reposo solo verifica pulsador de arranque
     if (digitalRead(arranque) == HIGH) Estado = ciclo;
     break;
  default:
     break;
  }
}

void serialEvent() {          // Función que atiende los mensajes (comunicación)
char dato;
if (Serial.available()) dato = Serial.read();       //lee caracter
if (dato=='C'){
    Serial.println (contador);
   }
else
   if (dato=='T'){
      total = 0;
   while (Serial.available())
      {
      dato = Serial.read();
      if (dato == '\r') break;
      total = total * 10 + dato - 48;
      }
       if ((total<4000)&&(total>10))
           t = total;
          Serial.println (t);
      }
}
```

Utilizamos la función **serialEvent()** del entorno Arduino, que la capa de abstracción de Arduino hace "transparente" al usuario, creando un buffer y recibiendo los datos por la UART hasta que se consulta por la existencia de datos al final del loop. Tiene varios inconvenientes pero es lo que ofrece el objeto Serial de Arduino. Entre otras cosas obliga a recorrer el loop para que se ejecute serialEvent (a menos que se llame a esta función).
La alternativa anterior supone que el mensaje completo ya está disponible en el buffer de recepción, pero no siempre es así. En ese caso habrá un error en la recepción de la consigna numérica. Una alternativa en la recepción de los mensajes del puerto serie es no salir de la rutina hasta que no se recibe el delimitador de final ("\r"). El problema de esta solución es que el control del programa queda suspendido hasta completar la recepción del mensaje.

```
void serialEvent() {          // Función que atiende los mensajes (comunicación)
char dato;
if (Serial.available()) dato = Serial.read();        //lee caracter
if (dato=='C')
   {
    Serial.println (contador);
   }
else
   if (dato=='T')
      {
      total = 0;
   while (1) // queda en lazo hasta recibir \r
      {
       if (Serial.available())
          {
          dato = Serial.read();
          if (dato == '\r') break;
          total = total * 10 + dato - 48;
          }
      }
       if ((total<4000)&&(total>10))
           t = total;
          Serial.println (t);
      }
}
```

Ejemplo 4: Oscilador "supervisado por **puerto serie**", con buffer de recepción y procesamiento de mensajes.

Una mejor solución es crear nuestro propio buffer de recepción (array), que se irá llenando hasta que se reciba el delimitador '\r'. Recién entonces pasaremos a una función para interpretar el mensaje, ya con todos los caracteres disponibles en dicho buffer. En este caso puede usarse también funciones de la stdlib como **atoi** o **atol**, para convertir la cadena de caracteres numéricos en un número entero.

```
/*
 * Ejercicio basado en el ejemplo SerialEvent de Arduino
  http://www.arduino.cc/en/Tutorial/SerialEvent
*/
#define arranque 11           // Es PB3 en UNO o Nano (ver pines en figura 3
#define parada 12             // Es PB4
#define salida 13             // Es PB5 y Led de placa UNO o Nano

enum tEstado{ciclo, reposo}; // Variable enumerada para representar Estado ciclo o reposo
enum tEstado Estado=ciclo;
int contador=0;
int t;

String Comando = "";         // buffer para almacenar el mensaje
bool comandoListo = false;   // flag para mensaje completo (cuando llega el caracter '\r')

void setup() {
  pinMode(salida, OUTPUT);
  pinMode(arranque, INPUT);
  pinMode(parada, INPUT);
  Serial.begin(9600);
  Comando.reserve(20);        //reserva un espacio de 20 caracteres para recibir mensaje
  t=500;
}

void loop()
{

switch (Estado){
  case ciclo:                                 // En estado activo genera pulso…
```

```
      contador++;
      digitalWrite(salida, HIGH);
      delay (t);
      digitalWrite(salida, LOW);
      delay(t);
      if (digitalRead(parada) == HIGH) Estado = reposo;      // … y verifica pulsador de parada
      break;
   case reposo:                                  // En reposo solo verifica pulsador de arranque
      if (digitalRead(arranque) == HIGH) Estado = ciclo;
      break;
   default:
      break;
   }
if (comandoListo) interpreta();
}

/*
  SerialEvent se ejecuta cuando se ha recibido uno o más caracteres (que quedan almacenados en
un buffer). No es una interrupción, sino que se verifica en el loop*/

void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    Comando += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
    if (inChar == '\r') {
      comandoListo = true;
    }
  }
}

/* interpreta()
  Esta rutina explora los caracteres almacenados en el buffer Comando. El primer carácter,
Comando[0], es para nuestro "protocolo" el identificador de función, que puede ser el carácter
'C' o el 'T'. En el caso de 'C' no hay caracteres posteriores. En el caso de 'T' vienen a
continuación los caracteres correspondientes a la consigna, expresados como una cadena
dedimal, que debe ser convertida a binario por el método visto anteriormente, o utilizando
directamente las funciones atoi o atol, disponibles en la biblioteca stdlib.h (en el entorno
Arduino ya está incluida)
*/

void interpreta(){
uint16_t aux16;
switch (Comando[0])
    {
      case 'C':
          Serial.print("Contador=");
          Serial.println(contador);
          break;
      case 'T':
          Serial.print("Tiempo=");
          aux16=atoi(&Comando[1]); //convierte a entero la cadena de caracteres guardadas en
                                   //Comando, pero empezando en el segundo carácter
                                   //(Comando[1])
          if (aux16<1000) t=aux16;
          Serial.println(t);
          break;
      default:
          break;
    }
        Comando = "";
        comandoListo = false;
}
```

Esta última solución, aunque no es óptima, es la que vamos a emplear en el Proyecto Integrador de Electrónica. En una próxima asignatura se manejará de forma más eficiente la comunicación y los periféricos en general.

Actividades a realizar (preparatorias para resolver el proyecto):

1. **Ensayo de los ejemplos.**
   Una manera de familiarizarse con el entorno Arduino es probar ejemplos. En este caso sugerimos copiar y pegar en el editor de Arduino los ejemplos vistos, y ensayar, ya sea sobre una placa Arduino o en un simulador. Hay muchos simuladores en línea. En caso de optar por el simulador Proteus, un procedimiento posible es el siguiente:

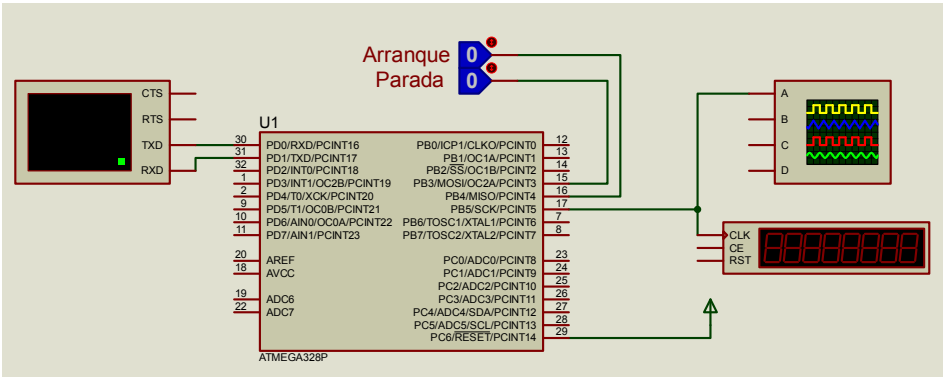


Figura 9. Circuito de simulación en Proteus

Debe colocarse el componente Atmega328P, el instrumento *Virtual Terminal*, el *Oscilloscope*, el instrumento *Counter Timer* (cambiar en propiedades a modo contador) y los componentes Logic Toggle como se indica en la figura 9.
Abrir las propiedades del Atmega328 (*doble click*) y colocar las propiedades indicadas (ver cada campo). (también es posible utiliza,en vez del Atmega328, placas Arduino u otros uCs)

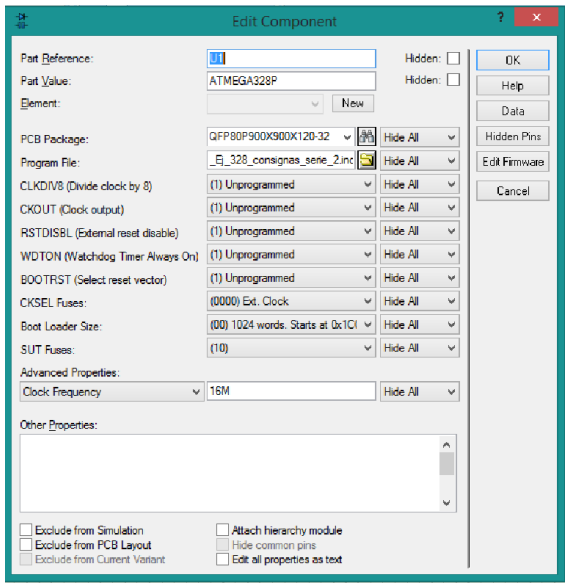


Figura 10. Ventana de propiedades del Atmega328P

En el campo “Program File” debe colocarse el archivo compilado por Arduino, que puede ser el .HEX o el .ELF. Dicho archivo puede localizarse en la ventana de mensajes de Arduino una vez compilado

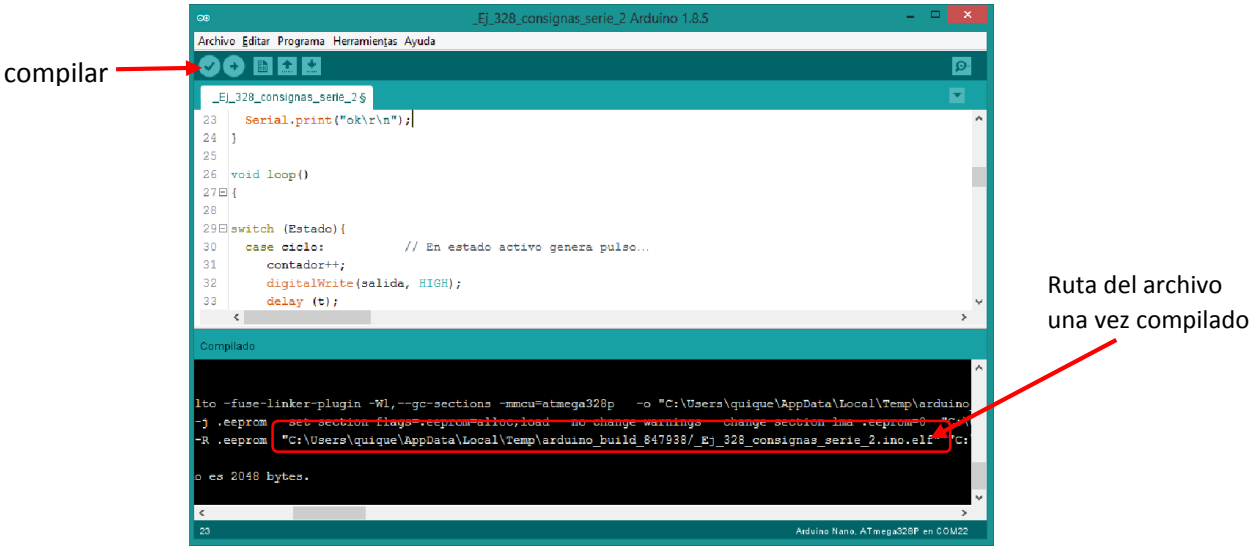


Figura 11. Entorno arduino

2. **Modificación o ampliación de los ejemplos**
   A los ejemplos anteriores introdúzcale sus propias modificaciones. Por ejemplo:
   - Incorporar comando ‘Z’ que pone a 0 el contador de pulsos
   - Utilizar tiempos distintos para el encendido y apagado de la salida, ejemplo TA y TB, que puedan modificarse por consignas ‘Annn\r’ y ‘Bnnn\r’. Es decir, deben realizar la misma función que los pines de entrada arranque y parada
   - Incorporar arranque y parada remotos, por ejemplo mediante los comandos ‘A’ y ‘P’.