UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Grado en Ingeniería informática

BLOQUE PRÁCTICO 2
Unidad Práctica 4

Sistemas Empotrados y de Tiempo Real

Antonio Carlos Domínguez Brito
Primer Semestre

Dafnis Jesús Santana Hernández
Renato Martín Sosa García

Contenido

OBJETIVO DE LA PRÁCTICA

El objetivo de esta unidad práctica es la integración del microcontrolador Atmel ATmega328P con diferentes tipos de sensores con el objeto de desarrollar un sistema empotrado con una funcionalidad específica.

En concreto, en esta práctica vamos a trabajar con el kit de motores RD03 de la marca Devantech. El objetivo es el diseño, desarrollo, prueba y despliegue de una librería para el control/monitorización del kit de motores RD03 de Devantech. La librería tiene que proporcionar una API en la que además de los comandos propios el kit de motores proporcionado por el fabricante permita también un control de velocidad de los mismos. Desarrollo de prototipo en el que pueda supervisarse el sistema a través del puerto serie desde un computador personal.

Además, existen objetivos comunes a todas las prácticas:

1. Debe implementarse una librería que permita reutilizar lo desarrollado con facilidad. Se proporcionará un repositorio git para el almacenamiento compartido de código en github.com.

2. Implementación de un reloj de tiempo real utilizando el Timer 1 para la medida de tiempo. Ello implica la implementación de algún mecanismo de establecimiento del tiempo actual utilizando el formato “dd/mm/aa hh:mm:ss” o similar, así como la conversión de dicho tiempo y los ticks de contabilizados por dicho timer.

3. Uso del almacenamiento de datos en memoria EEPROM.

4. Integración de display alfanumérico LCD05 de Devantech.

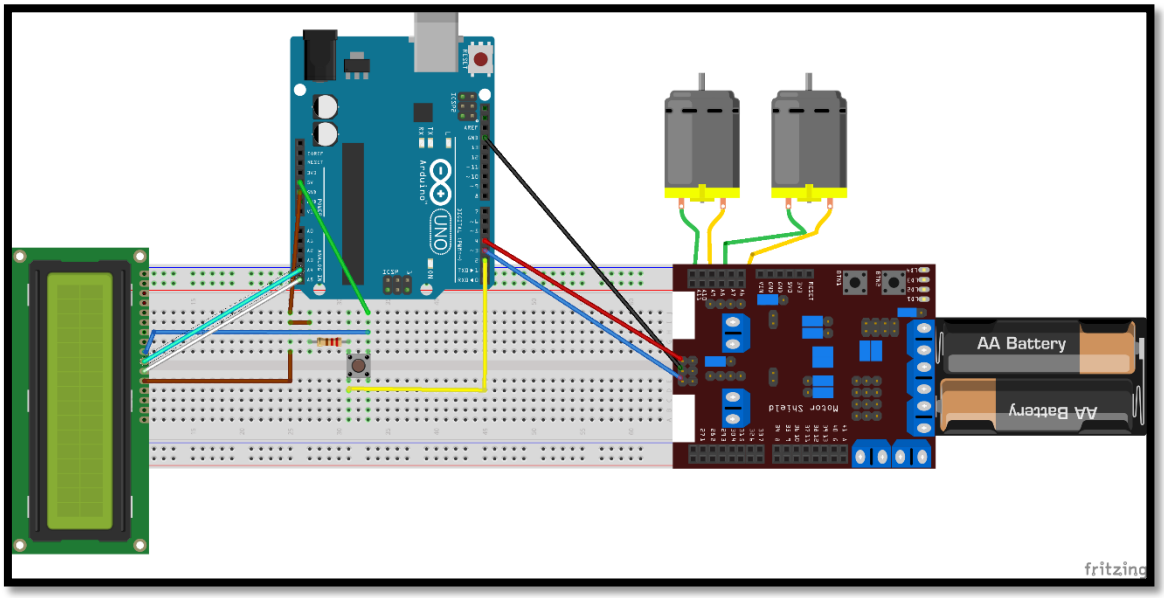
5. Uso de modos dormido.

6. Opcionalmente es posible utilizar módulos de comunicación inalámbrica XBee para la comunicación con el sistema.

MONTAJE HARDWARE

Para la práctica, hemos realizado el montaje de los motores, junto con la placa Arduino UNO utilizada en las demás prácticas, y un LCD para mostrar diferentes datos. Además, hemos utilizado una fuente de alimentación a 24V para alimentar a los motores (en la imagen se ha sustituido por pilas, solo a modo de informar de la existencia de una fuente de alimentación).

Hemos añadido un pulsador para despertar al Arduino del modo dormido.



REALIZACIÓN PRÁCTICA

Hemos utilizado la memoria EEPROM para almacenar la hora deseada, recuperarla o sustituirla por otra nueva, para lo cual hemos utilizado la librería EEPROM.h.

Para la gestión de la fecha y la hora, hemos utilizado la librería Time.h.

Mediante el comando duerme, el programa entra en modo dormido hasta ser despertado mediante el uso del pulsador. Cada segundo, el reloj Watchdog despertará al arduino para actualizar los valores del reloj y lo volverá al modo dormido. Para la gestión del modo dormido hemos utilizado la librería avr/sleep.h mientras que para el uso del Watchdog se hizo uso de la librería avr/wdt.h.

```
void duerme(){
  if(!wD){
    dormido = true ;
    stop();
    lcd.noBacklight();
  }

  attachInterrupt(0, pin2Interrupt, LOW);
  delay(100);

  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  sleep_mode();
  /* The program will continue from here. */
  /* First thing to do is disable sleep. */
  sleep_disable();
  refreshTime();
}
```

Configuración del Timer

Configuramos el Timer1, para que provoque una interrupción cada 50 ms.

```
TCCR1A = 0;// set entire TCCR1A register to 0
TCCR1B = 0;// same for TCCR1B
TCNT1  = 0;//initialize counter value to 0
TCCR1B |=  (1<<(WGM12));
OCR1A = 3124; // cada 50 ms
TCCR1B |= (1<<(CS12)); //prescaler 256
TIMSK1 |= (1<<(OCIE1A));
```

Configuración del Watchdog

```
void configureWatchdog() {
  cli();                // disable system interrupts during watchdog configuration
  wdt_reset();             // reset the watchdog timer
  WDTCSR |= (1<<WDCE) | (1<<WDE);   // follow unlocking procedure at the bottom of page 51 on the datasheet
  WDTCSR = 1<<WDP1 | 1<<WDP2;     // 1 seconds - Page 55 of the datasheet
  WDTCSR |= _BV(WDIE);        // Enable the WD interrupt (note no reset)
  sei();              // enable interrupts again, it's cool now
  wdt_reset();
}
```

Comunicación con los motores

Para la comunicación con los motores, hemos utilizado la librería SoftwareSerial.h, mediante la cual se permite configurar cualquier pin para realizar comunicación Serial. En este caso, utilizamos los pines 2 y 3 del arduino.

```
SoftwareSerial motors = SoftwareSerial(0x02, 0x03);    // Creates a serial port for the motors
```

Se han implementado las funciones necesarias para controlar los motores, utilizando los comandos proporcionados por el fabricante, de forma que la llamada a dichas funciones,

escriben el comando determinado mediante serial.

```
void setSpeed1(int velocidad){

  motors.write(CMD);
  motors.write(SET_SPEED1);
  motors.write(velocidad);
}

void setSpeed2(int velocidad){

  motors.write(CMD);
  motors.write(SET_SPEED2);
  motors.write(velocidad);
}


void setMode(int mode){

  motors.write(CMD);
  motors.write(SET_MOD);
  motors.write(mode);
}
```

```
int getSpeed1(){

  int speed=0;

  motors.write(CMD);
  motors.write(GET_SPEED1);
  while(motors.available() < 1){}
    speed = motors.read();
  return speed;
}

int getSpeed2(){

  int speed=0;
  motors.write(CMD);
  motors.write(GET_SPEED2);
  while(motors.available() < 1){}
    speed = motors.read();
  return speed;
}
```

Lectura de encoders y cálculo de la velocidad

La lectura de los encoders se realiza continuamente dentro del loop pero el momento exacto en el que se realiza queda registrado en las variables “registro1” para el motor 1 y “registro2” para el motor 2. Posteriormente, estos valores serán empleados para el cálculo de la velocidad.

La lectura en sí se realiza leyendo los 4 bytes de encoders que se obtienen al solicitar los encoders de uno de los motores, y después los combino en una variable int32.

```
int32_t getEncoder1(){

  motors.write(CMD);
  motors.write(GET_ENC1);

  while(motors.available() < 4){}

  encla = motors.read();
  enclb = motors.read();
  enclc = motors.read();
  encld = motors.read();

  en1 = (static_cast<uint32_t>(encla) << 24) +
  (static_cast<uint32_t>(enclb)<<16) +
  (static_cast<uint32_t>(enclc)<<8) +
  static_cast<uint32_t>(encld);
  return en1;
}

int32_t getEncoder2(){


  motors.write(CMD);
  motors.write(GET_ENC2);

  while(motors.available() < 4){}

  enc2a = motors.read();
  enc2b = motors.read();
  enc2c = motors.read();
  enc2d = motors.read();

  en2 = (static_cast<uint32_t>(enc2a) << 24) +
  (static_cast<uint32_t>(enc2b)<<16) +
  (static_cast<uint32_t>(enc2c)<<8) +
  static_cast<uint32_t>(enc2d);

  //en2=(enc2a << 24) | (enc2b <<16 ) | (enc2c <<8) | enc2d;

  return en2;
}

void leerEncoders(){

  en1 = getEncoder1();
  cli();
  registro1 = contador + TCNT1;
  sei();

  en2 = getEncoder2();
  cli();
  registro2 = contador + TCNT1;
  sei();
```

Una vez realiza la lectura, se lleva a cabo el cálculo de la velocidad, suponiendo primero que va hacia delante, y después que va hacia atrás. En el caso de ir hacia delante, comprobamos que,

si el encoder actual es mayor que el anterior, en ese caso los ticks avanzados hacia delante serán encoderactual-encoderanterior.

Sin embargo, si el encoder actual es menor que el encoder anterior, se habrá producido un desbordamiento, y por tanto los ticks avanzados hacia delante serán máximo-minimo+enconderAnterior-encoderActual.

```
if(en1>=enAnt1){

  tf1 = en1 - enAnt1;
}else{

  tf1 = maximum - minimum + en1 - enAnt1;
}
if(en2>=enAnt2){

  tf2 = en2 - enAnt2;
}else{

  tf2 = maximum - minimum + en2 - enAnt2;
}
```

A continuación, se realizan los cálculos suponiendo que va hacia atrás. En este caso, comprobamos que, si el encoder actual es menor que el anterior, en ese caso los ticks avanzados hacia atrás serán encoderAnterior-encoderActual.

Sin embargo, si el encoder actual es mayor que el anterior, se habrá producido un desbordamiento, y por tanto los ticks avanzados hacia delante serán máximo-minimo+encoderAnterior-encoderActual.

```
if(en1<=enAnt1){

  tb1 = enAnt1 - en1;
}else{

  tb1= maximum - minimum + enAnt1 - en1;
}
if(en2<=enAnt2){

  tb2 = enAnt2 - en2;
}else{

  tb2 = maximum - minimum + enAnt2 - en2;
}
```

Después de calcular ambos sentidos, nos quedamos con el menor valor entre ambos.

```
//OBTENER TICKS POR PERIODO
if(tb1<tf1){

  tk1 = -tb1;
}else{

  tk1 = tf1;
}
if(tb2<tf2){

  tk2 = -tb2;
}else{

  tk2 = tf2;
}
```

Finalmente, calculamos la velocidad km/h mediante la fórmula velocidad1=((tk1*2*M_PI/TICSV)/T)*RADIO, donde T es el tiempo transcurrido entre lecturas de encoders. Para concluir el cálculo de la velocidad, actualizamos los valores de los encoders para el siguiente cálculo.

```
T1 = registro1  - registro1Ant ;
T1 *= SECS_PER_TIMER1_TICK;
T2 = registro2  - registro2Ant ;
T2 *= SECS_PER_TIMER1_TICK;

velocidad1 = ( (tk1*2*M_PI/TICSV) / T1) * RADIO * 3.6;
velocidad2 = ( (tk2*2*M_PI/TICSV) / T2) * RADIO * 3.6;

registro1Ant = registro1;
registro2Ant = registro2;
en1Ant = en1;
en2Ant = en2;
}
```

Funcionamiento del programa

Al arrancar el programa, se nos pedirá que proporcionemos una fecha y hora en formato “dd/mm/yyyy hh:mm:ss”. Una vez recibido correctamente, el programa quedará a la espera de la introducción de alguno de los siguientes comandos:

- duerme: El Arduino entrará en sleep mode y solo se continuará con la ejecución del programa al pulsar el botón.
- velocidad n: Establece n como velocidad objetivo de ambos motores.
- speedX n: establece n como velocidad objetivo del motor x.
- back: invierte el sentido de ambos motores
- stop: detiene ambos motores y habilita la función resume.
- resume: reestablece la velocidad de los motores previa al stop.
- reset: reestablece la fecha a la última introducida.
- guardarFecha: permite reintroducir una nueva fecha y hora.

En todo momento se mostrará por el lcd la fecha y hora y la velocidad de ambos motores.

Como cierre de la práctica, hemos realizado un pequeño programa en Python para el manejo de los motores desde la consola de Python. En dicho código deberemos sustituir “COM11” por

el puerto al que esté conectado nuestro Arduino.

```
arduino = serial.Serial('COM11', 9600)
```

Ambos ficheros podrán encontrarse en el siguiente enlace:

https://github.com/DaffSH/SETR-DR03