TOBY FOOD

Alimentador Automático para Mascotas

Autores: Priscilla Rojas

Carrera: Análisis de Sistemas

Materia: Prácticas Profesionalizantes | 1º Año

PP1

2025

Informe Técnico

Introducción

El proyecto TobyFood surge con el objetivo de diseñar un sistema automatizado capaz de administrar alimento y agua a las mascotas de manera programada y eficiente. Este dispositivo busca mejorar la calidad de vida tanto del animal como del propietario, garantizando una alimentación controlada incluso en ausencia del usuario.

El sistema combina componentes electrónicos como sensores, actuadores y módulos de control, integrados en una placa Arduino Uno, para automatizar la dosificación de alimento sólido y el suministro de agua, manteniendo parámetros definidos por el usuario mediante un teclado (Keypad) y una pantalla LCD Display.

Objetivos

- Desarrollar un prototipo funcional de alimentador automático controlado por Arduino Uno.
- Implementar sensores y actuadores que garanticen un funcionamiento seguro y preciso.
- Incorporar almacenamiento de datos mediante memoria EEPROM para conservar configuraciones.
- Facilitar la interacción usuario-sistema mediante una interfaz LCD y teclado numérico.

Funcionamiento

Descripción General

El sistema TobyFood opera de forma automática o manual. El usuario define el intervalo de tiempo entre comidas y la cantidad de alimento en gramos, los cuales se guardan en la memoria EEPROM. El microcontrolador controla la apertura del compartimento mediante un servo motor, mientras que una celda de carga (Load Cell) con módulo HX711 mide el peso dispensado para garantizar precisión. Además, el sistema gestiona el nivel de agua mediante un sensor ultrasónico (Ultrasonic sensor) y una boya que activa o corta el flujo de agua controlado por una válvula solenoide. Un

sensor infrarrojo supervisa el depósito de alimento para evitar el funcionamiento en vacío.

Funciones principales

- **Dispensado automático de alimento:** El sistema sirve comida automáticamente cada cierto intervalo de horas, configurado por el usuario.
- **Pesaje en tiempo real:** La balanza integrada mide la cantidad servida para asegurar la dosis correcta.
- **Control de agua automático:** Si la boya detecta bajo nivel, la válvula se abre hasta completar el recipiente.
- **Alertas visuales:**
  - "Depósito bajo": indica falta de alimento.
  - "Depósito de agua bajo": muestra que el tanque está casi vacío.
  - "Error balanza": advierte fallas en la celda de carga.
- **Configuración guardada en memoria:** Los valores de cantidad e intervalo de tiempo permanecen grabados, aunque el sistema se apague.
- **Ahorro de energía:** La pantalla LCD se apaga automáticamente tras 30 segundos de inactividad.

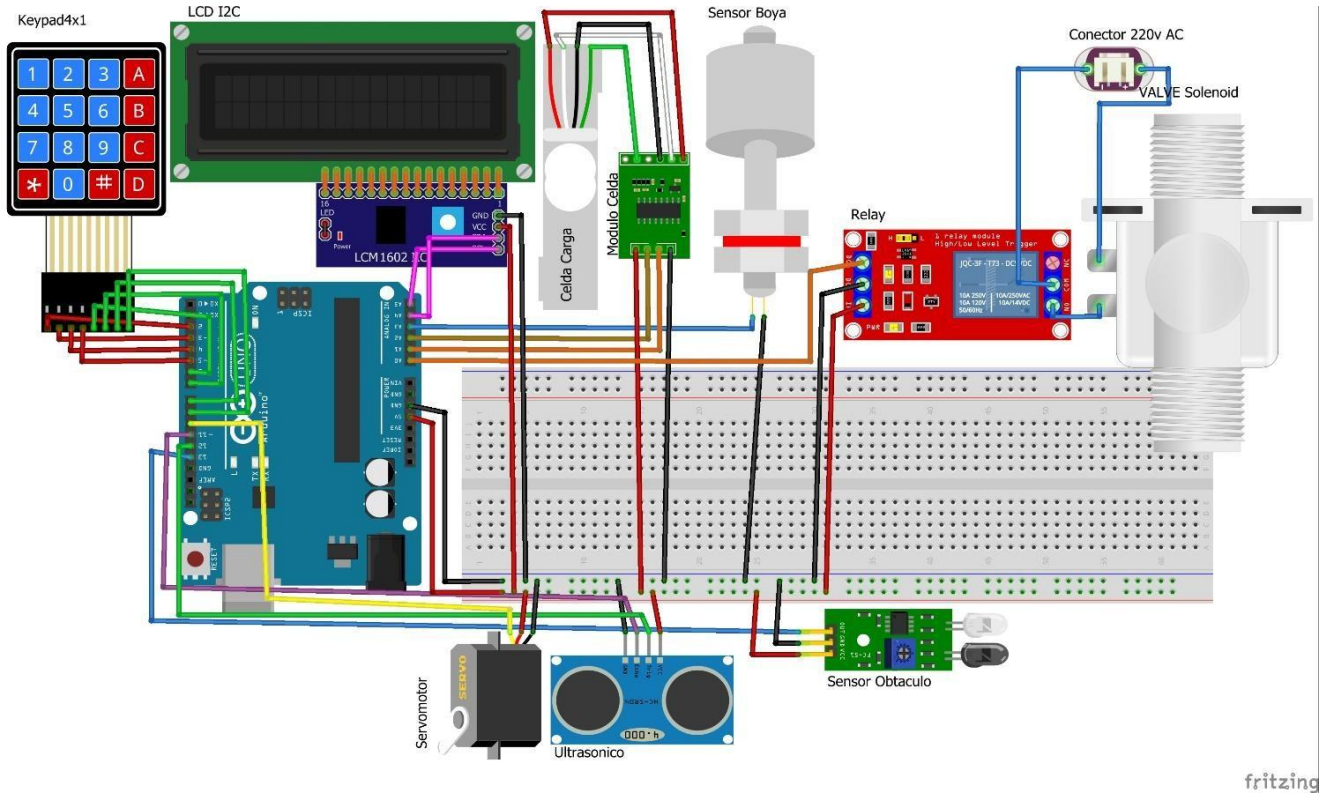
Interfaz de usuario (teclado)

Tecla	Función
A	Confirmar / Aceptar.
B	Cancelar o Borrar.
D	Iniciar manualmente el servido de comida.
#	Editar la configuración (cantidad de comida e intervalo en horas).
*****	Mostrar la configuración actual guardada.
0–9	Ingresar valores numéricos en configuración.

Componentes Utilizados

Componente	Función
Arduino UNO	Microcontrolador principal del sistema
LCD Display 16x2 (I2C)	Visualización de mensajes e información del sistema
Teclado 4x4 (Keypad)	Ingreso de datos por parte del usuario
Servo motor	Control de apertura del compartimento de comida.
Sensor infrarrojo	Detecta si el depósito de alimento está bajo.
Sensor ultrasonido (HC-SR04)	Mide el nivel del agua en el depósito.
Sensor de boya	Controla el nivel del bebedero.
Módulo HX711 + Celda de carga	Medición del peso de alimento dispensado.
Electroválvula 12V	Permite el llenado automático del recipiente de agua.
EEPROM interna	Guarda la configuración.

Circuito electrónico



Conexionado

1. Módulo HX711 — Celda de carga

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
DT	A1	Entrada digital	Recibe la señal de datos amplificada desde el HX711.	Conectado a la salida DT del módulo.
SCK	A2	Salida digital	Envía pulsos de reloj para sincronizar la transmisión.	Controlada por el Arduino.
VCC	5V	Alimentación	Proporciona tensión estable al módulo.	Requiere 5V regulados.
GND	GND	Tierra	Referencia eléctrica común del circuito.	Compartida con Arduino y celda.

2. Electroválvula 220 V (controlada por relé)

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
IN (módulo relé)	A0	Salida digital	Activa/desactiva el relé que alimenta la válvula.	Señal de control desde Arduino.
COM	220 V fase (entrada)	Potencia	Punto común del relé.	Conecta la línea de alimentación.
NO	220 V salida (a válvula)	Potencia	Envía corriente a la válvula al activarse.	Usa cableado con aislamiento.
VCC	5V	Alimentación	Energiza la bobina del relé.	Módulo compatible con 5 V.
GND	GND	Tierra	Referencia lógica común.	Conectada a GND de Arduino.

3. Sensor de boya (nivel de agua)

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
Signal	A3	Entrada digital (INPUT_PULLUP)	Detecta el estado de nivel (alto o bajo).	Cierra circuito al detectar agua.
GND	GND	Tierra	Retorno eléctrico del sensor.	Requiere resistor pull-up interno.

4. Servo motor (compuerta de alimento)

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
Signal	D10	Salida PWM	Controla la posición angular del servo.	Se maneja con la librería Servo.h.
VCC	5V	Alimentación	Proporciona potencia al motor.	Requiere 5V regulados.

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
GND	GND común	Tierra	Referencia eléctrica compartida.	Conectado también a GND de Arduino.

5. Sensor ultrasónico HC-SR04

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
TRIG	D12	Salida digital	Envía pulso ultrasónico de 10 µs.	Controlado por digitalWrite().
ECHO	D11	Entrada digital	Recibe eco del pulso reflejado.	Se mide duración para calcular distancia.
VCC	5V	Alimentación	Proporciona energía al sensor.	Consumo bajo (~15 mA).
GND	GND	Tierra	Referencia común.	Conexión indispensable para medición estable.

6. Sensor infrarrojo (nivel del depósito de alimento)

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
OUT	D13	Entrada digital	Detecta la presencia o ausencia de alimento.	Lógica inversa (LOW = sin alimento).
CC	5V	Alimentación	Alimenta el emisor y receptor IR.	Estable a 5 V.
GND	GND	Tierra	Retorno eléctrico.	Compartido con Arduino.

7. Teclado matricial 4x4

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
FILAS (R1–R4)	D9, D8, D7, D6	Entradas digitales	Detectan fila presionada.	Gestionadas por librería Keypad.h.
COLUMNAS (C1–C4)	D5, D4, D3, D2	Entradas digitales	Detectan columna presionada.	Usan multiplexado interno.

8. Pantalla LCD 16x2 con módulo I2C

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
SDA	A4	Comunicación I2C	Línea de datos serial.	Dirección predeterminada: 0x27.
SCL	A5	Comunicación I2C	Línea de reloj serial.	Utiliza librería LiquidCrystal_I2C.
VCC	5V	Alimentación	Fuente de energía de la pantalla.	5 V regulados.
GND	GND	Tierra	Retorno eléctrico común.	Conexión compartida.

9. Fuente de alimentación general

Conexión	Pin Arduino	Tipo	Función técnica	Observaciones
5 V	5V	Alimentación	Provee energía lógica.	Fuente regulada y estable.
GND	GND	Tierra	Referencia eléctrica común.	Unifica todos los retornos del sistema.

Código

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <EEPROM.h>
#include <Servo.h>
#include "HX711.h"

#define DT_PIN A1     // DT de HX711 -> pin A1
#define SCK_PIN A2    // SCK de HX711 -> pin A2

HX711 celda;
LiquidCrystal_I2C lcd(0x27, 16, 2);

// TECLADO
const byte FILAS = 4;
const byte COLUMNAS = 4;
char arrayPins[FILAS][COLUMNAS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
// Pines del keypad
byte arrayFilas[] = {9, 8, 7, 6};
byte arrayColumnas[] = {5, 4, 3, 2};


Keypad teclado = Keypad(makeKeymap(arrayPins), arrayFilas, arrayColumnas, FILAS, COLUMNAS);

// ACTUADORES / SENSORES
const int pinServo = 10;       // PWM para servo
const int trigPin = 12;        //pin trigger, ENVIA EL PULSO
const int echoPin = 11;        //pin echo, RECIBE EL PULSO
const int pinInfrarojo = 13;   // Sensor infrarrojo
const int pinValvula = A0;     // Control válvula
const int pinBoya = A3;        // Sensor boya




// Configuración
unsigned int intervaloHoras;
unsigned int cantidadGramos;

Servo miServo; // Creamos el objeto servo para controlar el servomotor
```

```
bool servir = false;
bool modificarConfig = false;
float escalaCelda = 1.0;
//const float KNOWN_WEIGHT = 100.0; // AJUSTAR según peso de calibración (g)
unsigned long ultimaComida = 0;
const unsigned long tiempoDeInactividad = 20000;
unsigned long ultimaActividad = 0;
bool menuActive = false;
float distancia;
float tiempo;

const int ADDR_INTERVALO = 0; // Posicion en memoria
const int ADDR_CANTIDAD = 10;
const int ADDR_ESCALA = 20;

int leerNumero(const char* mensaje) {
  String numerosIngresados = "";
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(mensaje);
  lcd.setCursor(0,1);
  Serial.println("Tecla presionada");
  while (true) {
    char teclaPresionada = teclado.getKey();
    if (teclaPresionada) {
      if (teclaPresionada >= '0' && teclaPresionada <= '9') {
        numerosIngresados += teclaPresionada;
        lcd.print(teclaPresionada);
      } else if (teclaPresionada == 'A') {
        if (numerosIngresados.length() > 0) {
          return numerosIngresados.toInt();
        }
      } else if (teclaPresionada == 'B') {
        if (numerosIngresados.length() > 0) {
          numerosIngresados.remove(numerosIngresados.length()-1);
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print(mensaje);
          lcd.setCursor(0,1);
          lcd.print(numerosIngresados);
        }
      }
    }
  }
}

// --- funciones de peso / servir
void servirComida(int pesoDeseado, float pesoActual){
  unsigned long t0 = millis();
```

```
  while (!celda.is_ready()) {
    if (millis() - t0 > 2000) break;
    delay(10);
  }
  if (!celda.is_ready()) { //VErificamos si la celda esta lista para usarse.
    lcd.clear();
    lcd.print("Error balanza");
    Serial.println("No se puede servir: HX711 no listo");
    delay(1500);
    return;
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("En proceso");
  Serial.println("Sirviendo");
  miServo.write(19);
  // delay(500);

  unsigned long tStart = millis();
  float pesoFormula = pesoDeseado - 50;

  while(pesoActual < pesoFormula){
    //Verifica que no sirva por mas de 30s
    if (millis() - tStart > 30000UL) { //Provar sino sacar
      Serial.println("Timeout sirviendo: detener.");
      lcd.clear();
      lcd.print("Verificar Deposito");
      break;
    }

    pesoActual = celda.get_units(10);
    Serial.print("Pesando... ");
    Serial.println(pesoActual);
    if (pesoActual >= pesoFormula) break;
    delay(100);
  }

  miServo.write(-5); // Regresa al origen
  delay(1000);
  lcd.clear();
  lcd.print("Comida Lista");
  Serial.print("Comida Lista");
  delay(4000);
  lcd.clear();
  servir = false;
}

void verificarPeso(){
```

```
  float pesoActual = celda.get_units(10); // promedio de 10 lecturas
  Serial.print("Peso actual: ");
  Serial.println(pesoActual);
  if (pesoActual < cantidadGramos) {
    Serial.println("Sirviendo...");
    servirComida(cantidadGramos, pesoActual);
  } else {
    Serial.println("Plato completo.");
  }
}

void configuracion(){
  intervaloHoras = leerNumero("Intervalo (hs):");
  cantidadGramos = leerNumero("Cantidad (g):");
  EEPROM.put(ADDR_INTERVALO, intervaloHoras);
  EEPROM.put(ADDR_CANTIDAD, cantidadGramos);
  lcd.clear();
  lcd.print("Guardado!");
  delay(1500);
  lcd.clear();
}

void setup() {
  Serial.begin(9600);
  Serial.println("Iniciando sistema");

  // Pines
  pinMode(pinValvula, OUTPUT);
  digitalWrite(pinValvula, LOW); // cerrar por defecto
  pinMode(pinBoya, INPUT_PULLUP);
  pinMode(pinInfrarojo, INPUT);
  pinMode(trigPin, OUTPUT); //de salida (envia pulso)
  digitalWrite(trigPin, LOW); //tener apagado el trigger
  pinMode(echoPin, INPUT);//de entrada (recibe el pulso)

  miServo.attach(pinServo);
  miServo.write(0);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Iniciando Pantalla");

  // Leer EEPROM (no sobrescribir aquí)
  EEPROM.get(ADDR_INTERVALO, intervaloHoras);
  EEPROM.get(ADDR_CANTIDAD, cantidadGramos);

  if (intervaloHoras <= 0 || intervaloHoras > 12 || cantidadGramos <= 0 || cantidadGramos >
500) {
```

```
  configuracion();
} else {
  Serial.print("Intervalo leido EEPROM: ");
  Serial.println(intervaloHoras);
  Serial.print("Cantidad leida EEPROM: ");
  Serial.println(cantidadGramos);
}

//CELDA DE CARGA
// HX711 inicialización
// Serial.println("Inicializando HX711 en pines A1 (DT) / A2 (SCK) ...");
// celda.begin(DT_PIN, SCK_PIN);

// unsigned long t0 = millis();
// while (!celda.is_ready()) {
//   if (millis() - t0 > 2000) break;
//   delay(10);
// }

// if (!celda.is_ready()) {
//   Serial.println("ATENCION: HX711 NO listo. Revisar cableado y alimentacion.");
// } else {
//   Serial.println("HX711 listo. Tare y calibracion...");
//   celda.set_scale(); // temporal
//   celda.tare();
//   delay(500);
//   Serial.println("Coloca peso conocido (ej. 100g) en 10s...");
//   lcd.clear();
//   lcd.print("Calibrndo");
//   delay(10000);

//   const int N = 10;
//   float lecturaProm = celda.get_units(N);
//   Serial.print("Lectura promedio cruda: ");
//   Serial.println(lecturaProm, 6);

//   if (KNOWN_WEIGHT <= 0.0) {
//     Serial.println("KNOWN_WEIGHT no valido. Ajustarlo.");
//   } else if (lecturaProm <= 0.0) {
//     Serial.println("Lectura cruda invalida; calibracion fallida.");
//   }else {
//     escalaCelda = lecturaProm / KNOWN_WEIGHT;
//     Serial.print("Factor calibracion: ");
//     Serial.println(escalaCelda, 6);
//     celda.set_scale(escalaCelda);
//     EEPROM.put(ADDR_ESCALA, escalaCelda);
//     float peso_calculado = celda.get_units(N);
//     Serial.print("Peso calculado despues de set_scale: ");
//     Serial.println(peso_calculado, 0);
```

```
  //   }
  // }


  //CONFIGURACION FINAL CELDA
  //Despues de que la calibracion haya sido exitosa:
  // Primero verifico que tenga el valor de la escala guaradada (Descomentar):
  // EEPROM.get(ADDR_ESCALA, escalaCelda);
  // Serial.println("Escala guardada: ");
  // Serial.println(escalaCelda);


  //Si lo de arriba no me da error, borrar o comentar la calibracion y solo dejar lo abajo
activo
  //(descomenatar lo de abajo):
  Serial.println("Probando Balanza...");
  celda.begin(DT_PIN, SCK_PIN);
  EEPROM.get(ADDR_ESCALA, escalaCelda);
  celda.set_scale(escalaCelda);
  celda.tare();

  ultimaActividad = millis();
  ultimaComida = millis();

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Sistema Listo!");
  Serial.println("Sist. completamente Config.");
  delay(3000);
  lcd.clear();
  lcd.noBacklight();
}

// loop principal: control de riego/servir/autodiagnóstico
void loop() {
  //Servir comida de forma AUTOMATICA
  unsigned long tiempoActual = millis();
  unsigned long intervaloMs = (unsigned long)intervaloHoras * 60000UL; //Minutos
  // unsigned long intervaloMs = (unsigned long)intervaloHoras * 3600000UL; //Horas

  if (intervaloMs > 0 && tiempoActual - ultimaComida >= intervaloMs) {
    float pesoActual = celda.get_units(10); // Leer peso actual del plato
    if (pesoActual < cantidadGramos) verificarPeso();
    ultimaComida = tiempoActual; // Actualizar la última comida
  }

  char teclaPresionada = teclado.getKey();

  if (teclaPresionada) {
```

```
  lcd.backlight();
  ultimaActividad = millis();
}

// CONFIGURACION TECLADO
if (teclaPresionada == 'D') {
  menuActive = true;
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Desea servir comida: ");
  lcd.setCursor(0, 1);
  lcd.print("A:Si B:NO ");
  delay(200);
  servir = true;

} else if (teclaPresionada == 'A' && servir) {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Iniciando...");
  delay(300);
  verificarPeso();
  menuActive = false;
  ultimaComida = tiempoActual; // Actualizar la última comida
  lcd.clear();

} else if (teclaPresionada == 'A' && modificarConfig) {
  lcd.clear();
  configuracion();
  modificarConfig = false;
  menuActive = false;
  lcd.clear();

} else if (teclaPresionada == 'B' && (servir || modificarConfig)) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Saliendo");
  delay(800);
  lcd.clear();
  servir = false;
  modificarConfig = false;
  menuActive = false;

} else if (teclaPresionada == '*') {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Intervalo: " + String(intervaloHoras) + "hs");
  lcd.setCursor(0,1);
  lcd.print( "Cantidad: " + String(cantidadGramos) + "g");
  delay(2000);
```

```
    lcd.clear();

  } else if (teclaPresionada == '#') {
    menuActive = true;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Edit config");
    lcd.setCursor(0, 1);
    lcd.print("A:Si B:NO");
    modificarConfig = true;
  }


  //BOYA - controla valvula
  int estadoBoya = digitalRead(pinBoya);

  if (estadoBoya == LOW) {  // Agua faltante
    digitalWrite(pinValvula, HIGH); // Abre válvula
    if (!menuActive) {
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.backlight();
      lcd.print("Cargando Agua");
      delay(3000);
    }
  }else {
    digitalWrite(pinValvula, LOW);
  }

  // Sensor DEPOSITO: comida baja (infrarrojo)
  int sensorComidaBaja = digitalRead(pinInfrarojo);
  if (sensorComidaBaja == HIGH) {
    lcd.backlight();
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Deposito bajo");
    lcd.setCursor(0, 1);
    lcd.print("Cargar Comida");
    delay(3000);
    lcd.clear();
  }
  // agua_bajo();
  digitalWrite(trigPin, HIGH); //generar pulso
  delayMicroseconds (10); //tarda 10 microsegundos segun el datasheet
  digitalWrite(trigPin,LOW);
  tiempo = pulseIn(echoPin,HIGH); //pulseIn devuelve un vialor de tiempo en microsegundos,
cuanto tarda la señal en ir y volver
  distancia = tiempo / 58.77; //calcular la distancia usando la formula del datasheet
```

```
  if (distancia > 12){
    Serial.println(distancia);
    lcd.clear();
    lcd.backlight();
    lcd.setCursor(0,0);
    lcd.print("DEPOSITO BAJO");
    lcd.setCursor(0,1);
    lcd.print("Cargar Agua");
    delay(3000);
  }

   // Apagar backlight si inactividad (opcional)
  if (millis() - ultimaActividad > tiempoDeInactividad) {
    lcd.clear(); //Limpia la pantalla
    lcd.noBacklight(); // Apaga la luz de fondo
    menuActive = false;
  }
}
```

DATASHEETS

CELDA DE CARGA DE 20 KG

OKY3480-2

Descripción

El OKY3480-2 es un sensor de peso transductor que soporta cargas de compresión, tensión y flexión de hasta 20kg de peso y las convierte en una magnitud eléctrica proporcional a la carga. Un extremo del sensor se fija mediante tornillos mientras que el otro extremo quedara suspendido. Consiste en una barra de aluminio con agujeros roscados para montaje y 4 galgas extensiométricas unidas a la barra. La aplicación principal de las celdas de carga es la medición de peso en la industria de alimentos, medicamentos, transporte entre otras.

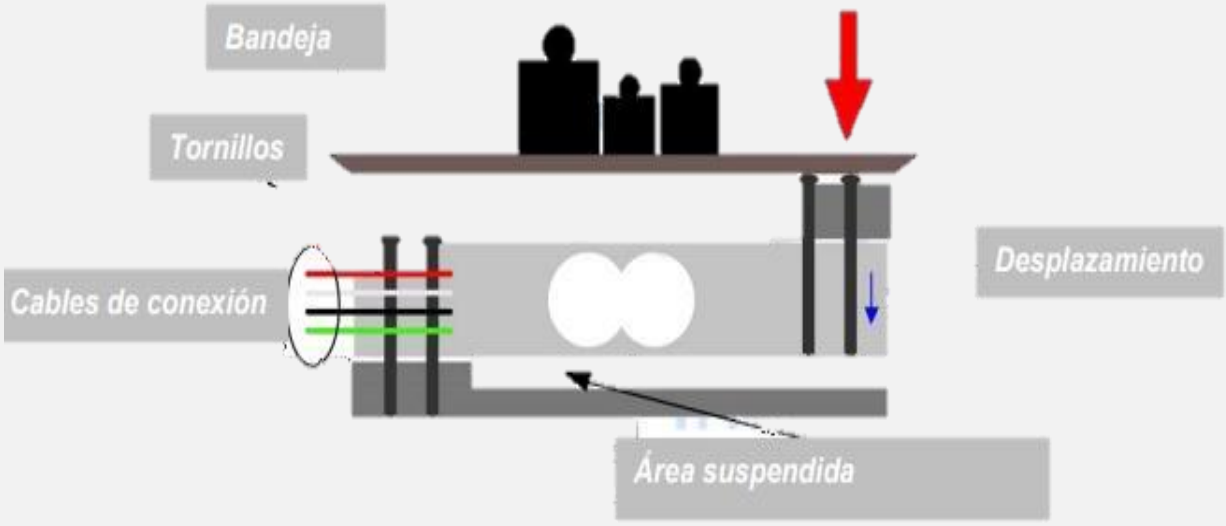
Especificaciones:

Parámetro	Descripción
Voltaje máximo de trabajo	5 ~ 10 VDC
Sensibilidad de salida	1.0 ± 0.1mV / V
Impedancia de entrada	1000 ± 50 Ω
Impedancia de salida	1000 ± 50 Ω
Carga nominal	20 KG

Error integral	0.05% F.S
Temperatura de salida nominal	≤ 0.15% F.S / 10°C
Deriva del punto cero	0.05% F.S (1 minute)
Variación del punto cero con la temperatura	0.2% F.S / 10°C
Salida cero	± 0.1mV / V
Capacidad de sobre carga	150% F.S
Rango de temperatura	-10°C a 50°C
Tamaño	80 mm x 13 mm x 13 mm
Peso	73 gramos

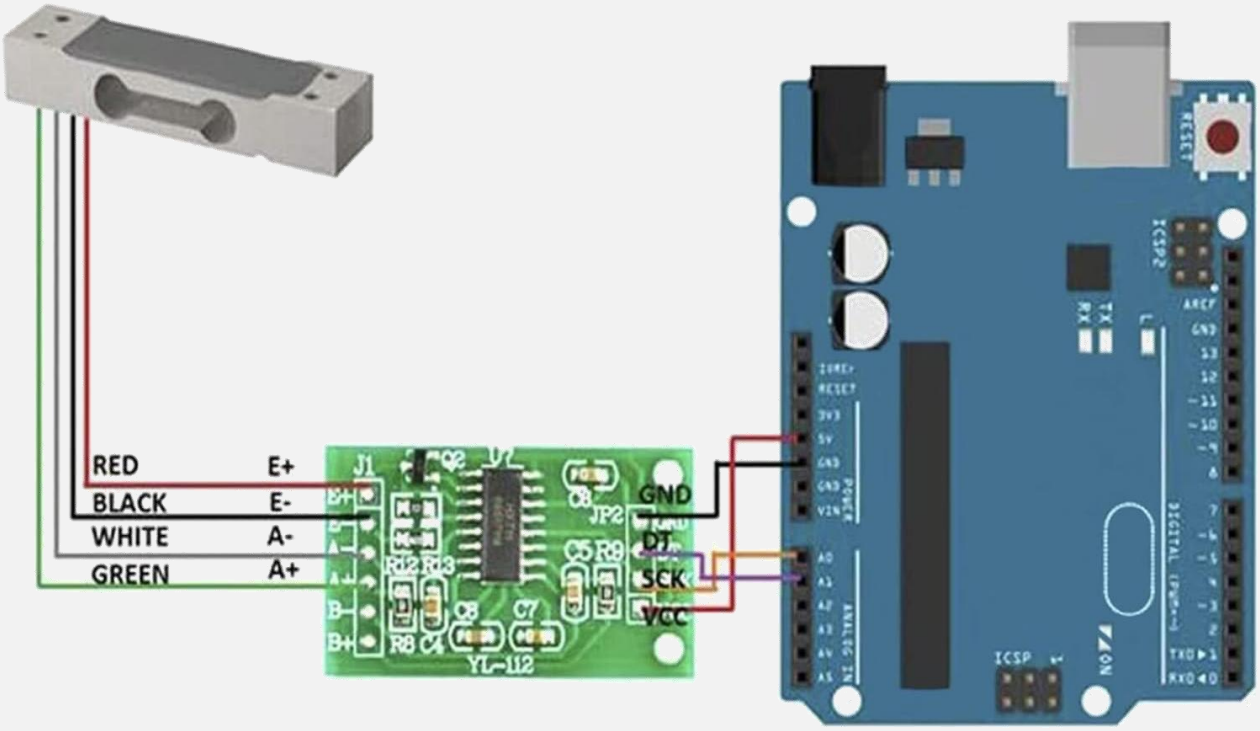
Características

- Cable de alimentación: Rojo +, Negro -
- Cable de salida voltaje: Verde +, Blanco -



Conexión de la celda en Arduino

Conecta los cables de la celda de carga a los pines E+ y E- del módulo HX711. Asegúrate de respetar la polaridad adecuada.
Conecta los cables de la celda de carga a los pines A+ y A- del módulo HX711.
Puedes realizar las conexiones de acuerdo al siguiente diagrama.



El módulo HX711. no viene incluido pero lo puedes adquirir de nuestra tienda virtual entrando al siguiente enlace:

https://www.agelectronica.com/detalle.php?p=OKY3478-2

Ejemplo de código en Arduino

Es necesario descargar la librería HX711: https://github.com/bogde/HX711
Recuerde importar esta librería al IDE de Arduino

```
#include "HX711.h" // Incluye la biblioteca HX711 para la comunicación con el sensor de peso

#define DOUT A1 // Define el pin A1 como DOUT para la señal de datos del HX711
#define CLK A0    // Define el pin A0 como CLK para la señal de reloj del HX711

HX711 balanza(DOUT, CLK); // Crea una instancia del objeto HX711 llamado balanza, utilizando los pines DOUT y CLK definidos
```

```
void setup() {
   Serial.begin(9600); // Inicia la comunicación serie a 9600 baudios
   Serial.print("Lectura del valor del ADC: ");
   Serial.println(balanza.read()); // Lee el valor actual del ADC y lo imprime en el monitor
serie
   Serial.println("No ponga ningun objeto sobre la balanza");
   Serial.println("Calculando...");
balanza.set_scale(); // Establece la escala a 1 por defecto
balanza.tare(20); // Realiza la tara, considerando el peso actual como cero (toma 20
lecturas)
   Serial.println("Coloque un peso conocido:");
}

void loop() {
   Serial.print("Valor de lectura: ");
   Serial.println(balanza.get_value(10), 0); // Lee el valor de la balanza promediando 10
lecturas y lo imprime en el monitor serie
   delay(100); // Espera 100 milisegundos antes de la siguiente lectura
}
```

Con el promedio de estos datos medidos calculamos el valor de la escala que usaremos, para esto usaremos la siguiente formula:

Escala= valor medido / peso real

```
#include "HX711.h" // Incluye la biblioteca HX711 para la comunicación con el sensor de
peso

#define DOUT A1 // Define el pin A1 como DOUT para la señal de datos del HX711
#define CLK A0    // Define el pin A0 como CLK para la señal de reloj del HX711

HX711 balanza(DOUT, CLK); // Crea una instancia del objeto HX711 llamado balanza,
utilizando los pines DOUT y CLK definidos

void setup() {
   Serial.begin(9600); // Inicia la comunicación serie a 9600 baudios

   Serial.print("Lectura del valor del ADC: ");
   Serial.println(balanza.read()); // Lee el valor actual del ADC y lo imprime en el monitor
serie

   Serial.println("No ponga ningun objeto sobre la balanza");
   Serial.println("Destarando...");
   Serial.println("...");
```

```
balanza.set_scale(439430.25); // Establece la escala utilizando el factor de calibración específico
  balanza.tare(20); // Realiza la tara tomando 20 lecturas para considerar el peso actual como cero
Serial.println("Listo para pesar");
}

void loop() {

Serial.print("Peso: ");
  Serial.print(balanza.get_units(20), 3); // Lee el peso promediando 20 lecturas y lo imprime en kg con 3 decimales
  Serial.println(" kg");
  delay(500); // Espera 500 milisegundos antes de la siguiente lectura
```

User Guide

I2C Serial Interface 1602 LCD Module

This is I2C interface 16x2 LCD display module, a high-quality 2 line 16 character LCD module with on-board contrast control adjustment, backlight and I2C communication interface. For Arduino beginners, no more cumbersome and complex LCD driver circuit connection. The real significance advantages of this I2C Serial LCD module will simplify the circuit connection, save some I/O pins on Arduino board, simplified firmware development with widely available Arduino library.

SKU: DSP-1182

Brief Data:

- Compatible with Arduino Board or other controller board with I2C bus.
- Display Type: Negative white on Blue backlight.
- I2C Address:0x38-0x3F (0x3F default)
- Supply voltage: 5V
- Interface: I2C to 4bits LCD data and control lines.
- Contrast Adjustment: built-in Potentiometer.
- Backlight Control: Firmware or jumper wire.
- Board Size: 80x36 mm.

Setting Up:

Hitachi's HD44780 based character LCD are very cheap and widely available, and is an essential part for any project that displays information. Using the LCD piggy-back board, desired data can be displayed on the LCD through the I2C bus. In principle, such backpacks are built around PCF8574 (from NXP) which is a general purpose bidirectional 8 bit I/O port expander that uses the I2C protocol. The PCF8574 is a silicon CMOS circuit provides general purpose remote I/O expansion (an 8-bit quasi-bidirectional) for most microcontroller families via the two-line bidirectional bus (I2C-bus). Note that most piggy-back modules are centered around PCF8574T (SO16 package of PCF8574 in DIP16 package) with a default slave address of 0x27. If your piggy-back board holds a PCF8574AT chip, then the default slave address will change to 0x3F. In short, if the piggy-back board is based on PCF8574T and the address connections (A0-A1-A2) are not bridged with solder it will have the slave address 0x27.

Address selection pads in the I2C-to-LCD piggy-back board.

Table 5. PCF8574A address map

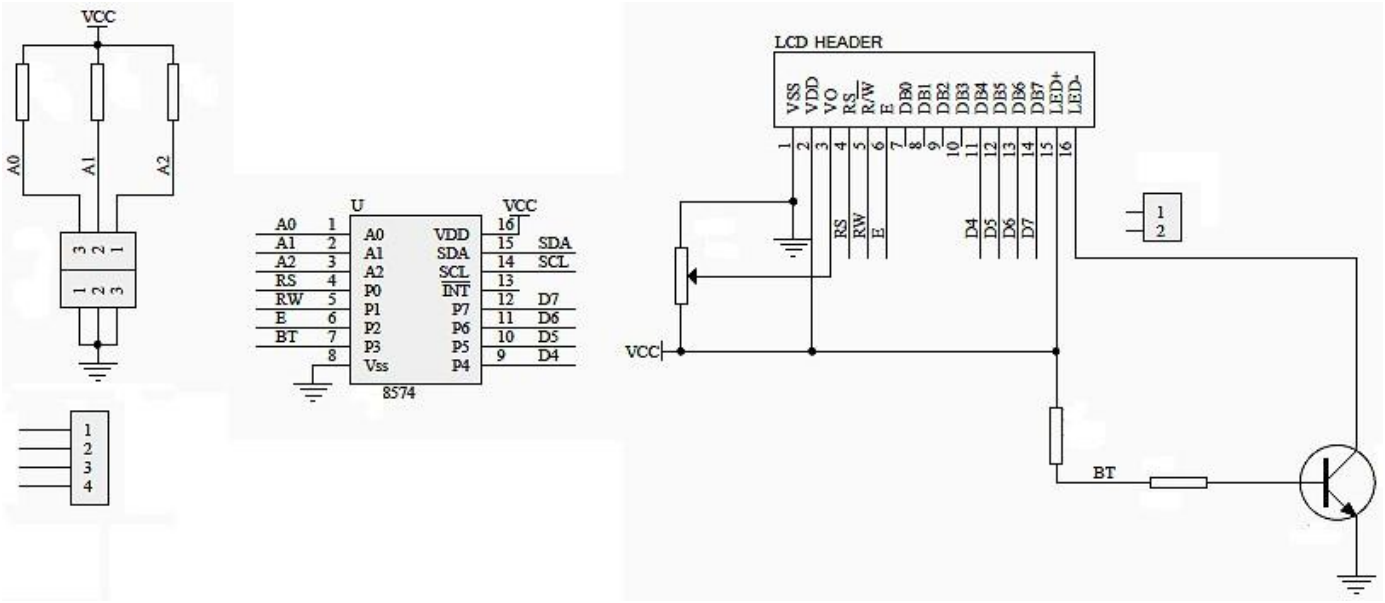
Pin connectivity			Address of PCF8574A								Address byte value		7-bit hexadecimal address without R/W
A2	A1	A0	A6	A5	A4	A3	A2	A1	A0	R/W	Write	Read	
V_{SS}	V_{SS}	V_{SS}	0	1	1	1	0	0	0	-	70h	71h	38h
V_{SS}	V_{SS}	V_{DD}	0	1	1	1	0	0	1	-	72h	73h	39h
V_{SS}	V_{DD}	V_{SS}	0	1	1	1	0	1	0	-	74h	75h	3Ah
V_{SS}	V_{DD}	V_{DD}	0	1	1	1	0	1	1	-	76h	77h	3Bh
V_{DD}	V_{SS}	V_{SS}	0	1	1	1	1	0	0	-	78h	79h	3Ch
V_{DD}	V_{SS}	V_{DD}	0	1	1	1	1	0	1	-	7Ah	7Bh	3Dh
V_{DD}	V_{DD}	V_{SS}	0	1	1	1	1	1	0	-	7Ch	7Dh	3Eh
V_{DD}	V_{DD}	V_{DD}	0	1	1	1	1	1	1	-	7Eh	7Fh	3Fh

Address Setting of PCD8574A (extract from PCF8574A data specs).

Note: When the pad A0~A2 is open, the pin is pull up to VDD. When the pin is solder shorted, it is pull down to VSS.

The default setting of this module is A0~A2 all open, so is pull up to VDD. The address is 3Fh in this case.

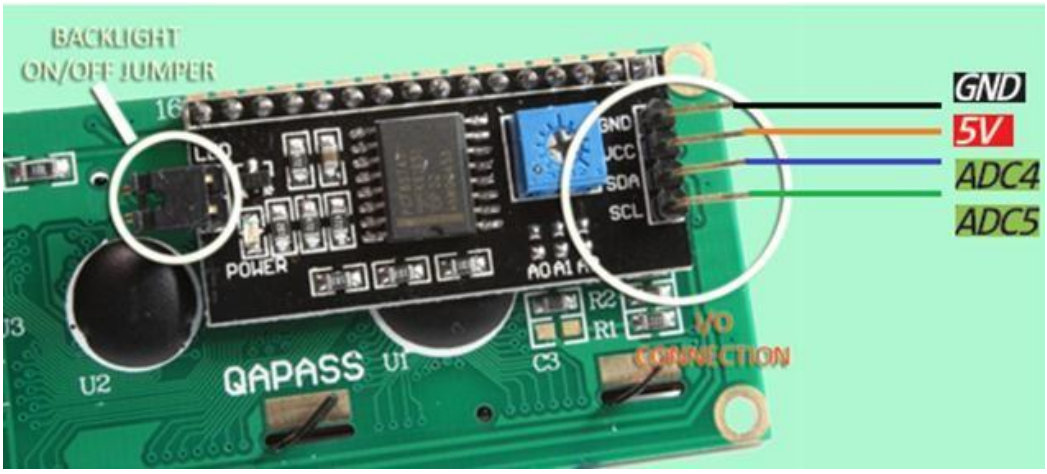
Reference circuit diagram of an Arduino-compatible LCD backpack is shown below. What follows next is information on how to use one of these inexpensive backpacks to interface with a microcontroller in ways it was exactly intended.



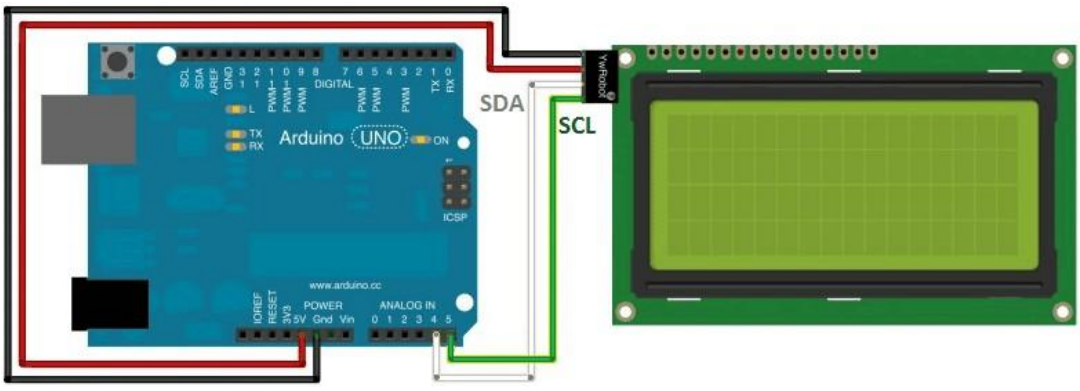
Reference circuit diagram of the I2C-to-LCD piggy-back board.

I2C LCD Display.

At first you need to solder the I2C-to-LCD piggy-back board to the 16-pins LCD module. Ensure that the I2C-to-LCD piggy-back board pins are straight and fit in the LCD module, then solder in the first pin while keeping the I2C-to- LCD piggy-back board in the same plane with the LCD module. Once you have finished the soldering work, get four jumper wires and connect the LCD module to your Arduino as per the instruction given below.



LCD display to Arduino wiring.



Arduino Setup

For this experiment it is necessary to download and install the "Arduino I2C LCD" library. First of all, rename the existing "LiquidCrystal" library folder in your Arduino libraries folder as a backup, and proceed to the rest of the process.

https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

Next, copy-paste this example sketch Listing-1 for the experiment into the blank code window, verify, and then upload.

Arduino Sketch Listing-1:

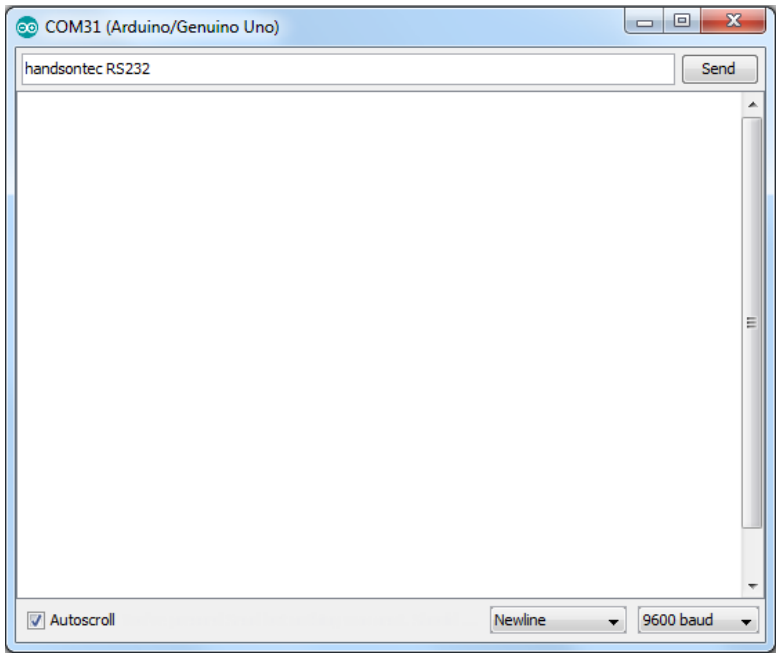
```
/*============================================================================
//  Author       : Handson Technology
//  Project      : I2C to LCD with Arduino Uno
//  Description : LCD with I2C Interface.
//  LiquidCrystal Library - I2C Serial to LCD
//  Source-Code : I2C LCD.ino
//============================================================================
*/

/*-----( Import needed libraries )-----*/
#include <Wire.h> // Comes with Arduino IDE
// Get the LCD I2C Library here:
// https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads
// Move any other LCD libraries to another folder or delete them
// See Library "Docs" folder for possible commands etc.

#include <LiquidCrystal_I2C.h>
/*-----( Declare Constants )-----*/
// set the LCD address to 0x3F for PCF8574AT with A0,A1,A0 address line open, default
setting.
// Set the pins on the I2C chip used for LCD connections:
//                    (addr, en,rw,rs,d4,d5,d6,d7,bl,blpol)
LiquidCrystal_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C
address

/*-----( Declare Variables )-----*/


void setup()    /*----( SETUP: RUNS ONCE )----*/
{
  Serial.begin(9600);  // Used to type in characters

  lcd.begin(20,4);          // initialize the lcd for 20 chars 4 lines, turn on
backlight

// ------- Quick 3 blinks of backlight  -------------
  for(int i = 0; i< 3; i++)
  {
    lcd.backlight();
    delay(250);
    lcd.noBacklight();
    delay(250);
  }
  lcd.backlight(); // finish with backlight on

//-------- Write characters on the display ------------------
  // NOTE: Cursor Position: Lines and Characters start at 0
  lcd.setCursor(3,0); //Start at character 4 on line 0
  lcd.print("Hello, world!");
  delay(1000);
  lcd.setCursor(2,1);
  lcd.print("From Handsontec  ");
```

```
  delay(1000);
  lcd.setCursor(0,2);
  lcd.print("20 by 4 Line Display");
  lcd.setCursor(0,3);
  delay(2000);
  lcd.print(" www.handsontec.com ");
  delay(8000);
  // Wait and then tell user they can start the Serial Monitor and type in characters
to
  // Display. (Set Serial Monitor option to "No Line Ending")
  lcd.setCursor(0,0); //Start at character 0 on line 0
  lcd.print("Start Serial Monitor");
  lcd.setCursor(0,1);
  lcd.print("Type char to display");


}/*--(end setup )---*/


void loop()    /*----( LOOP: RUNS CONSTANTLY )----*/
{
  {
    // when characters arrive over the serial port...
    if (Serial.available()) {
      // wait a bit for the entire message to arrive
      delay(100);
      // clear the screen
      lcd.clear();
      // read all the available characters
      while (Serial.available() > 0) {
        // display each character to the LCD
        lcd.write(Serial.read());
      }
    }
  }

}/* --(end main loop )-- */


/* ( THE END ) */
```

If you are 100% sure that everything is okay, but you don't see any characters on the display, try to adjust the contrast control pot of the backpack and set it a position where the characters are bright and the background does not have dirty boxes behind the characters. Following is a partial view of author's experiment with the above described code with 20x4 display module. Since the display used by the author is a very clear bright "black on yellow" type, it is very difficult to get a good catch due to polarization effects.

This sketch will also display character send from serial Monitor:

In Arduino IDE, go to “Tools” > “Serial Monitor”. Set the correct baud rate at 9600. Type the character on the top empty space and hit “SEND”.



The string of character will be displayed on the LCD module.

Resources:

- Handson Technology
- Complete Guide to Arduino LCD Interfacing (PDF)

PRODUCT SPECIFICATION

PCB JQC-T73

- **10A switching capability**
- **Small footprint**
- **Sealed type available**
- **Class B/F available**
- **Conform to RoHS,ELV directive**
- **Size : 19.2X15.4X15.4mm**

ORDERING CODE

JQC - 3FF - S - H 1 2 3	
1. Relay Model JQC-T73 2. S: sealed	3. Z: Form C H: Form A D: Form B

COIL DATA (at 20 °C)

Nominal Voltage (VDC)	3	5	6	9	12	18	24	48	0.36W
Coil Resistance (Ω± 10%)	25	69	100	225	400	900	1600	6400	
Rated Current (mA)	120	71.4	60	40	30	20	15	7.5	
Max Operate Voltage (VDC)	2.25	3.75	4.5	6.75	9	13.5	18	36	
Min Release Voltage (VDC)	0.15	0.25	0.3	0.45	0.6	0.9	1.2	2.4	
Max ApplicableVoltage	130% of nominal voltage at 70 °C 170% of nominal voltage at 23 °C								

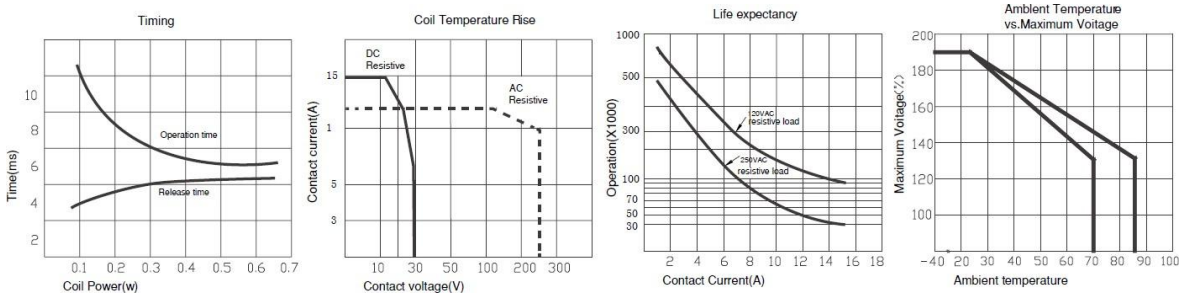
CONTACT DATA

Contact Form	1H/ 1Z
Contact Material	Silver Alloy
Load	Resistive load(COSΦ= 1)
Contact Ratings	10A 250vac 15A 125vac 10A 28vdc
Minimum load	100mA 5VDC
Max Switching Voltage	250VAC/ 30VDC
Max Switching Current	15A
Max Switching Power	2770VA/ 240W
Contact Resistance	100mΩMax at 6VDC 1A
Life Expectancy	Electrical : 100,000 Operations(at 30Operations/ minute)
	Mechanical : 10,000,000 Operations(at 300Operations/ minute)

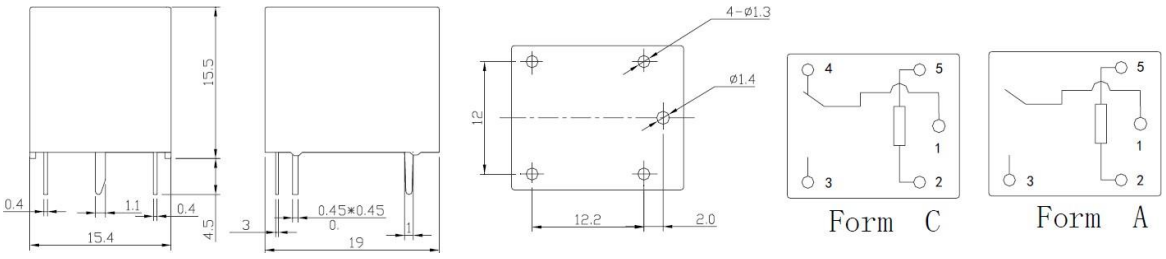
CHARACTERISTICS DATA

Insulation Resistance	100MΩMin at 500VDC
Dielectric Strength Between Open Contacts	750VAC(50/60Hz for one minute)
Between Contacts and coil	1500VAC(50/60Hz for one minute)
Operate Time	10ms
Release Time	5ms
Temperature Range	-40 °C to+85 °C
Shock Resistance	Operating Extremes: 10G
	Damage Limits: 100G
Vibration Resistance	10-55Hz, 1.5mm
Max. switching frequency	Mechanical: 18,000 operations/hr
	Electrical: 1,800 operations/hr
Humidity	40-85%
Weight	Approx 10g
Safety Standard	CQC UL SGS TUV

ENGINEERING DATA



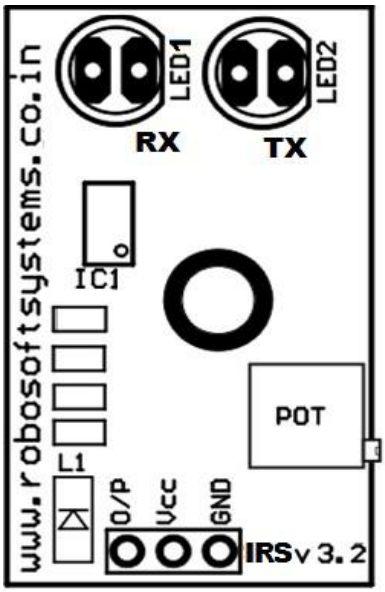
OVERALL AND MOUNTING DIMENSIONS



IR Sensor Based obstacle detection sensor module (Single)

IR Sensor - Single

General Description



The IR Sensor-Single is a general purpose proximity sensor. Here we use it for collision detection. The module consist of a IR emitter and IR receiver pair. The high precision IR receiver always detects a IR signal.

The module consists of 358 comparator IC. The output of sensor is high whenever it IR frequency and low otherwise. The on-board LED indicator helps user to check status of the sensor without using any additional hardware.

The power consumption of this module is low. It gives a digital output.

Pin Configuration

The figure to the right is a top view of the IR Sensor module. The following table gives its pin description.

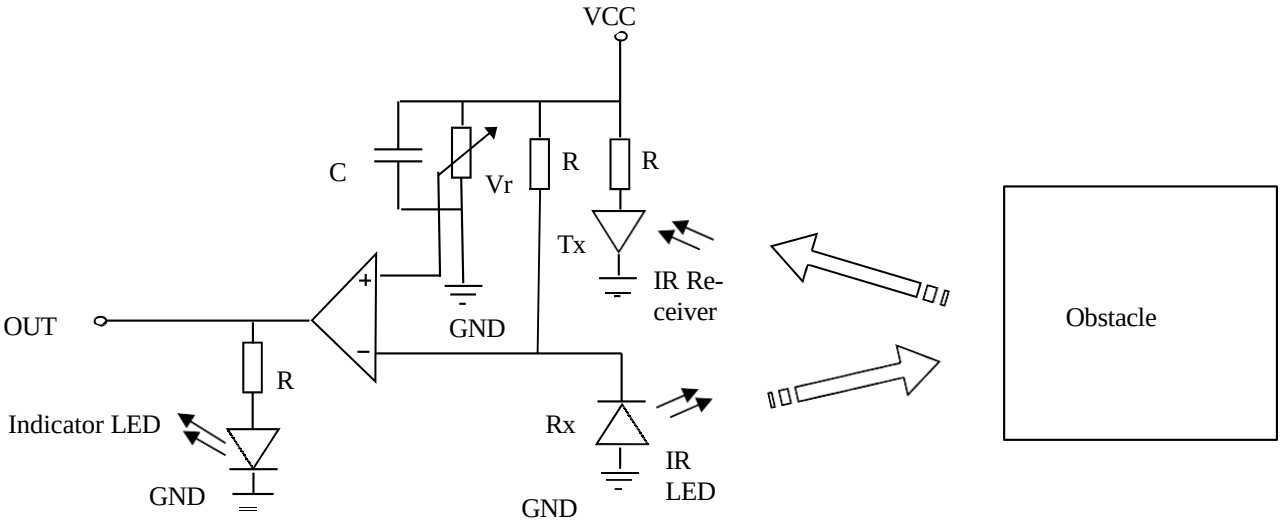
Pin No.	Connection	Description
1	Output	Digital Output (High or Low)
2	VCC	Connected to circuit supply
3	Ground	Connected to circuit ground

Application Ideas

- Obstacle detection
- Shaft encoder
- Fixed frequency detection

IR Sensor - Single

Functional Block Diagram /Schematic Diagram



IR Sensor - Single

Overview of Schematic

The sensitivity of the IR Sensor is tuned using the potentiometer. The potentiometer is tuneable in both the directions. Initially tune the potentiometer in clockwise direction such that the Indicator LED starts glowing. Once that is achieved, turn the potentiometer just enough in anti-clockwise direction to turn off the Indicator LED. At this point the sensitivity of the receiver is maximum. Thus, its sensing distance is maximum at this point. If the sensing distance (i.e., Sensitivity) of the receiver is needed to be reduced, then one can tune the potentiometer in the anti-clockwise direction from this point.

Further, if the orientation of both Tx and Rx LED's is parallel to each other, such that both are facing outwards, then their sensitivity is maximum. If they are moved away from each other, such that they are inclined to each other at their soldered end, then their sensitivity reduces.

Tuned sensitivity of the sensors is limited to the surroundings. Once tuned for a particular surrounding, they will work perfectly until the IR illumination conditions of that region nearly constant. For example, if the potentiometer is tuned inside room/building for maximum sensitivity and then taken out in open sunlight, its will require retuning, since sun's rays also contain Infrared (IR) frequencies, thus acting as a IR source (transmitter). This will disturb the receiver's sensing capacity. Hence it needs to be retuned to work perfectly in the new surroundings.

The output of IR receiver goes low when it receives IR signal. Hence the output pin is normally low because, though the IR LED is continuously transmitting, due to no obstacle, nothing is reflected back to the IR receiver. The indication LED is off. When an obstacle is encountered, the output of IR receiver goes low, IR signal is reflected from the obstacle surface. This drives the output of the comparator low. This output is connected to the cathode of the LED, which then turns ON.

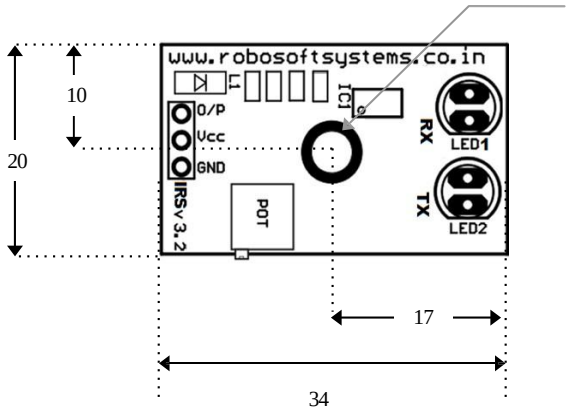
Note : All dimension in mm
Error of ±**Maximum Ratings**

Symbol	Quantity	Minimum	Typical	Maximum	Unit
o/p	Output Voltage	0	-	5	V
V_{CC}	Operating Voltage	4.5	5	5.5	V
GND	Ground Reference voltage	-	0	-	V

Pin Out Dimensions

R 2

IR Sensor - Single



Note : All dimension in mm
Error of ±5% is subjected because of component soldering

24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales

DESCRIPTION

Based on Avia Semiconductor's patented technology, HX711 is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

The input multiplexer selects either Channel A or B differential input to the low-noise programmable gain amplifier (PGA). Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of ±20mV or ±40mV respectively, when a 5V supply is connected to AVDD analog power supply pin. Channel B has a fixed gain of 32. On-chip power supply regulator eliminates the need for an external supply regulator to provide analog power for the ADC and the sensor. Clock input is flexible. It can be from an external clock source, a crystal, or the on-chip oscillator that does not require any external component. On-chip power-on-reset circuitry simplifies digital interface initialization.

There is no programming needed for the internal registers. All controls to the HX711 are through the pins.

FEATURES

- **Two selectable differential input channels**
- **On-chip active low noise PGA with selectable gain of 32, 64 and 128**
- **On-chip power supply regulator for load-cell and ADC analog power supply**
- **On-chip oscillator requiring no external component with optional external crystal**
- **On-chip power-on-reset**
- **Simple digital control and serial interface: pin-driven controls, no programming needed**
- **Selectable 10SPS or 80SPS output data rate**
- **Simultaneous 50 and 60Hz supply rejection**
- **Current consumption including on-chip analog power supply regulator: normal operation < 1.5mA, power down < 1uA**
- **Operation supply voltage range: 2.6 ~ 5.5V**
- **Operation temperature range: -40 ~ +85℃**
- **16 pin SOP-16 package**

APPLICATIONS

- **Weigh Scales**
- **Industrial Process Control**

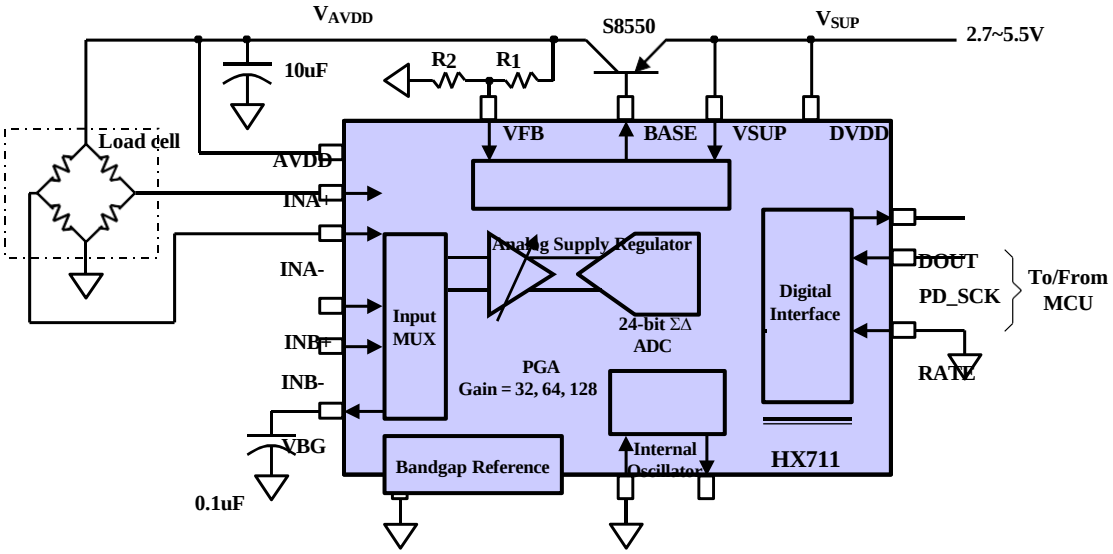
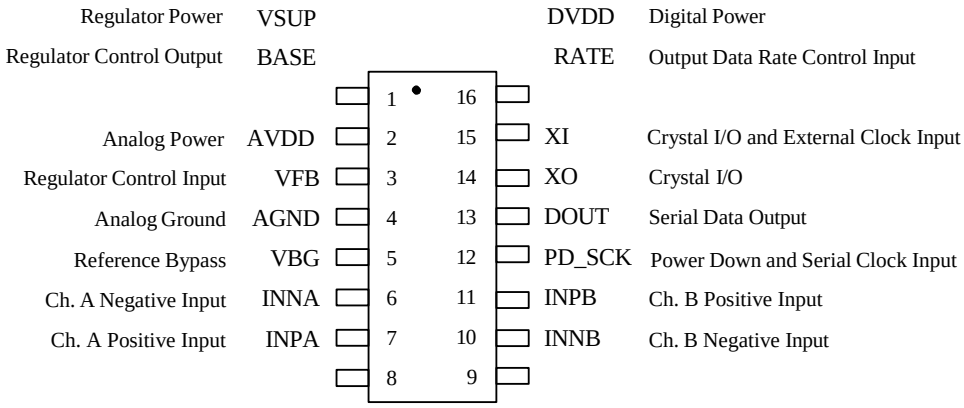


Fig. 1 Typical weigh scale application block diagram

Pin Description



SOP-16L Package

Pin #	Name	Function	Description
1	VSUP	Power	Regulator supply: 2.7 ~ 5.5V
2	BASE	Analog Output	Regulator control output（NC when not used）
3	AVDD	Power	Analog supply: 2.6 ~ 5.5V
4	VFB	Analog Input	Regulator control input（connect to AGND when not used）
5	AGND	Ground	Analog Ground
6	VBG	Analog Output	Reference bypass output
7	INA-	Analog Input	Channel A negative input
8	INA+	Analog Input	Channel A positive input
9	INB-	Analog Input	Channel B negative input
10	INB+	Analog Input	Channel B positive input
11	PD_SCK	Digital Input	Power down control (high active) and serial clock input
12	DOUT	Digital Output	Serial data output
13	XO	Digital I/O	Crystal I/O (NC when not used）
14	XI	Digital Input	Crystal I/O or external clock input, 0: use on-chip oscillator
15	RATE	Digital Input	Output data rate control, 0: 10Hz; 1: 80Hz
16	DVDD	Power	Digital supply: 2.6 ~ 5.5V

Table 1 Pin Description

KEY ELECTRICAL CHARACTERISTICS

Parameter	Notes	MIN	TYP	MAX	UNIT
Full scale differential input range	V(inp)-V(inn)		±0.5(AVDD/GAIN)		V
Common mode input		AGND+1.2		AVDD-1.3	V
Output data rate	Internal Oscillator, RATE = 0		10		Hz
	Internal Oscillator, RATE = DVDD		80		
	Crystal or external clock, RATE = 0		f_{clk}/1,105,920		
	Crystal or external clock, RATE = DVDD		f_{clk}/138,240		
Output data coding	2's complement	800000		7FFFFF	HEX
Output settling time (1)	RATE = 0		400		ms
	RATE = DVDD		50		
Input offset drift	Gain = 128		0.2		mV
	Gain = 64		0.4		
Input noise	Gain = 128, RATE = 0		50		nV(rms)
	Gain = 128, RATE = DVDD		90		
Temperature drift	Input offset（Gain = 128）		±6		nV/℃
	Gain（Gain = 128）		±5		ppm/℃
Input common mode rejection	Gain = 128, RATE = 0		100		dB
Power supply rejection	Gain = 128, RATE = 0		100		dB
Reference bypass（V_{BG}）			1.25		V
Crystal or external clock frequency		1	11.0592	20	MHz
Power supply voltage	DVDD	2.6		5.5	V
	AVDD, VSUP	2.6		5.5	
Analog supply current (including regulator)	Normal		1400		μA
	Power down		0.3		
Digital supply current	Normal		100		μA
	Power down		0.2		

（1）Settling time refers to the time from power up, reset, input channel change and gain change to valid stable output data.

Table 2 Key Electrical Characteristics

Analog Inputs

Channel A differential input is designed to interface directly with a bridge sensor's differential output. It can be programmed with a gain of 128 or 64. The large gains are needed to accommodate the small output signal from the sensor. When 5V supply is used at the AVDD pin, these gains correspond to a full-scale differential input voltage of ±20mV or ±40mV respectively.

Channel B differential input has a fixed gain of 32. The full-scale input voltage range is ±80mV, when 5V supply is used at the AVDD pin.

Power Supply Options

Digital power supply (DVDD) should be the same power supply as the MCU power supply.

When using internal analog supply regulator, the dropout voltage of the regulator depends on the external transistor used. The output voltage is equal to $V_{AVDD}=V_{BG}*(R1+R2)/R2$ (Fig.1). This voltage should be designed with a minimum of 100mV below VSUP voltage.

If the on-chip analog supply regulator is not used, the VSUP pin should be connected to either AVDD or DVDD, depending on which voltage is higher. Pin VFB should be connected to Ground and pin BASE becomes NC. The external 0.1uF bypass capacitor shown on Fig. 1 at the VBG output pin is then not needed.

Clock Source Options

By connecting pin XI to Ground, the on-chip oscillator is activated. The nominal output data rate when using the internal oscillator is 10 (RATE=0) or 80SPS (RATE=1).

If accurate output data rate is needed, crystal or external reference clock can be used. A crystal can be directly connected across XI and XO pins. An external clock can be connected to XI pin, through a 20pF ac coupled capacitor. This external clock is not required to be a square wave. It can come directly from the crystal output pin of the MCU chip, with amplitude as low as 150 mV.

When using a crystal or an external clock, the internal oscillator is automatically powered down.

Output Data Rate and Format

When using the on-chip oscillator, output data rate is typically 10 (RATE=0) or 80SPS (RATE=1).

When using external clock or crystal, output data rate is directly proportional to the clock or crystal frequency. Using 11.0592MHz clock or crystal results in an accurate 10 (RTE=0) or 80SPS (RATE=1) output data rate.

The output 24 bits of data is in 2's complement format. When input differential signal goes out of the 24-bit range, the output data will be saturated at 800000h (MIN) or 7FFFFFh (MAX), until the input signal comes back to the input range.

Serial Interface

Pin PD_SCK and DOUT are used for data retrieval, input selection, gain selection and power down controls.

When output data is not ready for retrieval, digital output pin DOUT is high. Serial clock input PD_SCK should be low. When DOUT goes to low, it indicates data is ready for retrieval. By applying 25~27 positive clock pulses at the PD_SCK pin, data is shifted out from the DOUT output pin. Each PD_SCK pulse shifts out one bit, starting with the MSB bit first, until all 24 bits are shifted out. The 25^{th} pulse at PD_SCK input will pull DOUT pin back to high (Fig.2).

Input and gain selection are controlled by the number of the input PD_SCK pulses (Table 3). PD_SCK clock pulses should not be less than 25 or more than 27 within one conversion period, to avoid causing serial communication error.

PD_SCK Pulses	Input channel	Gain
25	A	128
26	B	32
27	A	64

Table 3 Input Channel and Gain Selection

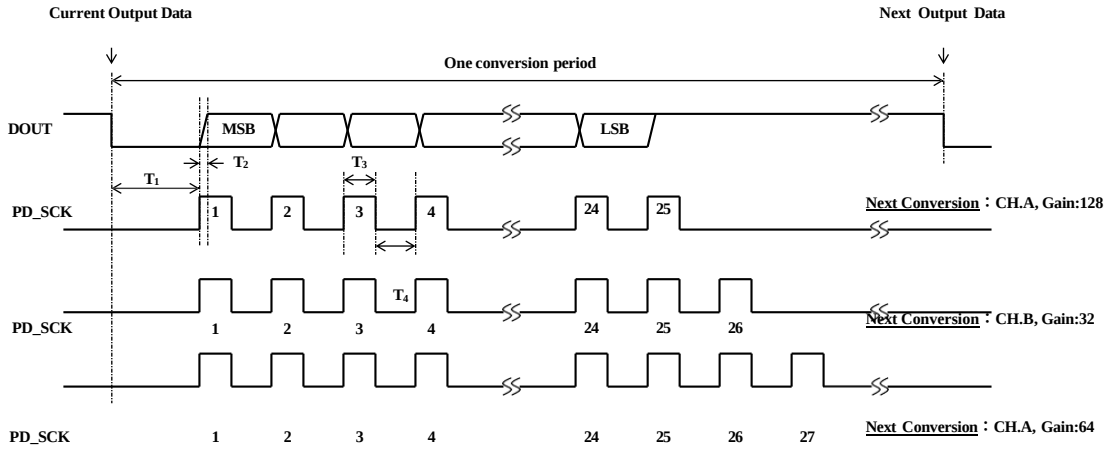


Fig.2 Data output, input and gain selection timing and control

Symbol	Note	MIN	TYP	MAX	Unit
T_1	DOUT falling edge to PD_SCK rising edge	0.1			μs
T_2	PD_SCK rising edge to DOUT data ready			0.1	μs
T_3	PD_SCK high time	0.2	1	50	μs
T_4	PD_SCK low time	0.2	1		μs

Reset and Power-Down

When chip is powered up, on-chip power on rest circuitry will reset the chip.

Pin PD_SCK input is used to power down the HX711. When PD_SCK Input is low, chip is in normal working mode.

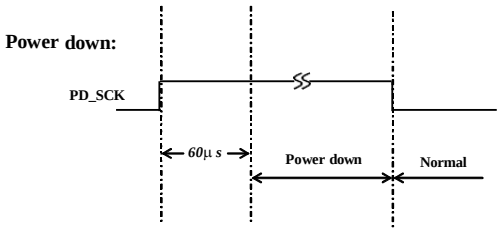


Fig.3 Power down control

When PD_SCK pin changes from low to high and stays at high for longer than 60μs, HX711 enters power down mode (Fig.3). When internal regulator is used for HX711 and the external transducer, both HX711 and the transducer will be powered down. When PD_SCK returns to low,

chip will return back to the setup conditions before power down and enter normal operation mode.

If PD_SCK pulse number is changed during the current conversion period, power down should be executed after current conversion period is completed. This is to ensure that the change is saved. When chip returns back to normal operation from power down, it will return to the set up conditions of the last change.

Application Example

Fig.1 is a typical weigh scale application using HX711. It uses on-chip oscillator (XI=0), 10Hz output data rate (RATE=0). A Single power supply （2.7～5.5V） comes directly from MCU power supply. Channel B can be used for battery level detection. The related circuitry is not shown on Fig. 1.

Reference PCB Board (Single Layer)

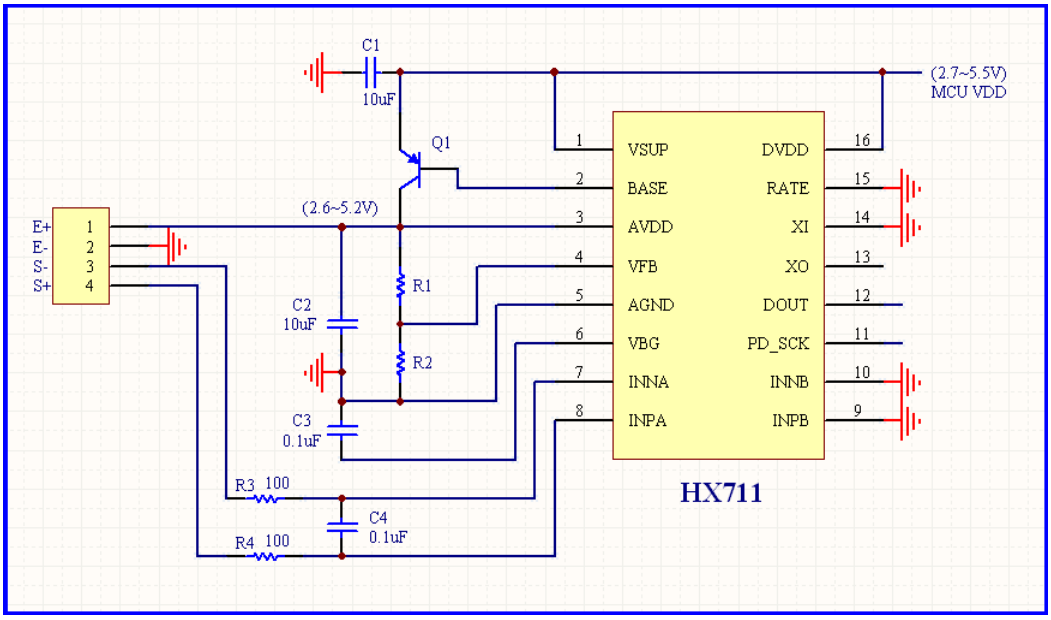


Fig.4 Reference PCB board schematic

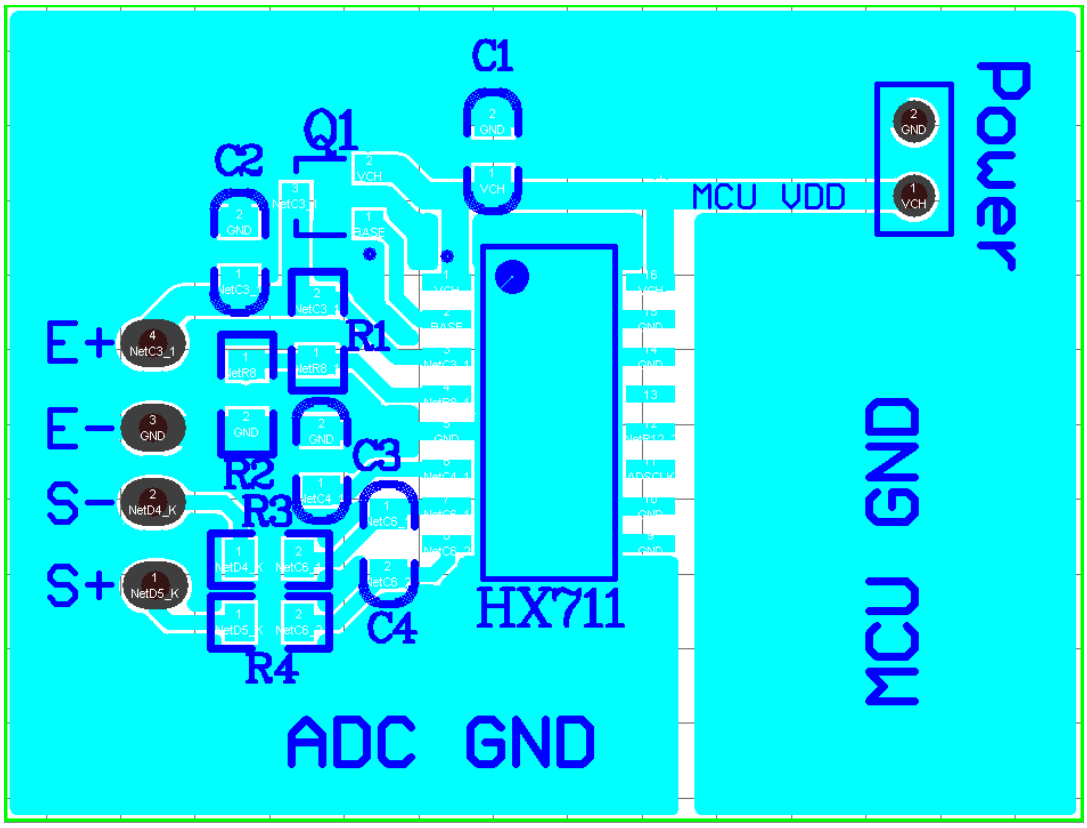


Fig.5 Reference PCB board layout

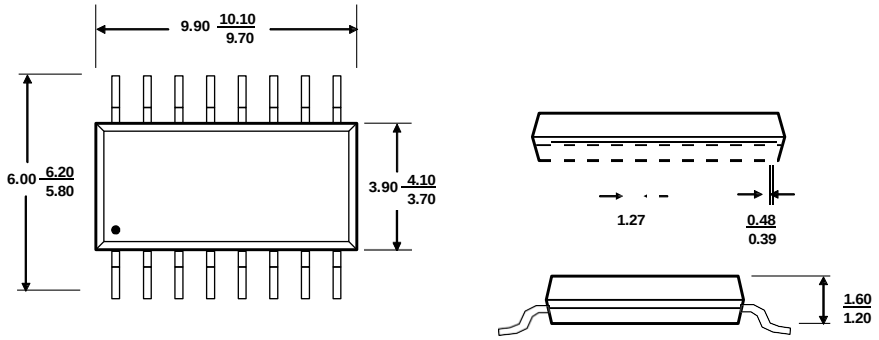
Reference Driver (Assembly)

```
/*------------------------------------------------------------------------------------------------
Call from ASM:      LCALL   ReaAD
Call from C:     extern unsigned long ReadAD(void);
                          .
                          .
                        unsigned long data;
                        data=ReadAD();
                          .
                          .
------------------------------------------------------------------------------------------------*/
PUBLIC          ReadAD
HX711ROM        segment  code
rseg            HX711ROM

sbit            ADDO = P1.5;
sbit            ADSK = P0.0;
/*-------------------------------------------------------------------------
 OUT:   R4, R5, R6, R7    R7=>LSB
-------------------------------------------------------------------------*/
ReadAD:
    CLR     ADSK            //AD Enable  (PD_SCK set low)
    SETB    ADDO            //Enable 51CPU I/0
    JB      ADDO,$          //AD conversion completed?
    MOV     R4,#24
ShiftOut:
    SETB    ADSK            //PD_SCK  set  high  (positive  pulse)
    NOP
    CLR     ADSK            //PD_SCK set low
    MOV     C,ADDO          //read on bit
    XCH     A,R7            //move data
    RLC     A
    XCH     A,R7
    XCH     A,R6
    RLC     A
    XCH     A,R6
    XCH     A,R5
    RLC     A
    XCH     A,R5
    DJNZ    R4,ShiftOut     //moved 24BIT?
    SETB    ADSK
    NOP
    CLR     ADSK
    RET
    END
```

Reference Driver（C）

```
//-----------------------------------------------------------------------------------------------------
sbit    ADDO = P1^5;
sbit    ADSK = P0^0;
unsigned long ReadCount(void){
  unsigned long Count;
  unsigned char i;
  ADDO=1;
  ADSK=0;
  Count=0;
  while(ADDO);
  for (i=0;i<24;i++){
    ADSK=1;
    Count=Count<<1;
    ADSK=0;
    if(ADDO) Count++;
  }
  ADSK=1;
  Count=Count^0x800000;
  ADSK=0;
  return(Count);
}
```

Package Dimensions



SOP-16L Package

Revision History

revision	record
1.0	initial version
2.0	change “Reset and Power-Down”

HC-SR04 Ultrasonic Sensor

Elijah J. Morgan

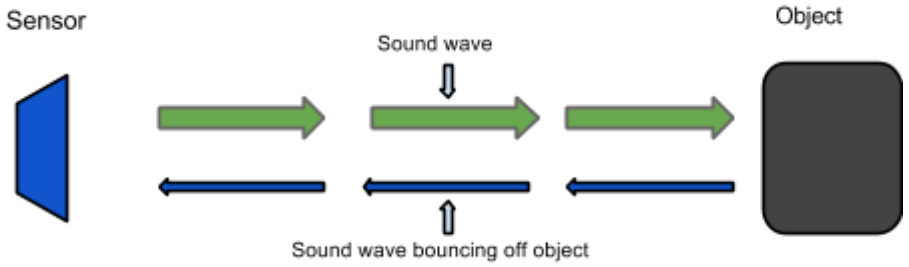
Nov. 16 2014

The purpose of this file is to explain how the HC-SR04 works. It will give a brief explanation of how ultrasonic sensors work in general. It will also explain how to wire the sensor up to a microcontroller and how to take/interpret readings. It will also discuss some sources of errors and bad readings.

1. **How Ultrasonic Sensors Work**
2. **HC-SR04 Specifications**
3. **Timing chart, Pin explanations and Taking Distance Measurements**
4. **Wiring HC-SR04 with a microcontroller**
5. **Errors and Bad Readings**

1. How Ultrasonic Sensors Work

Ultrasonic sensors use sound to determine the distance between the sensor and the closest object in its path. How do ultrasonic sensors do this? Ultrasonic sensors are essentially sound sensors, but they operate at a frequency above human hearing.



The sensor sends out a sound wave at a specific frequency. It then listens for that specific sound wave to bounce off of an object and come back (Figure 1). The sensor keeps track of the time between sending the sound wave and the sound wave returning. If you know how fast something is going and how long it is traveling you can find the distance traveled with equation 1.

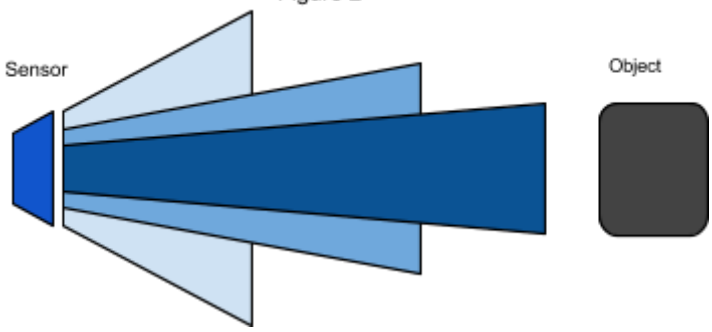
Equation 1. $d = v \times t$

The speed of sound can be calculated based on the a variety of atmospheric conditions, including temperature, humidity and pressure. Actually calculating the distance will be shown later on in this document.

It should be noted that ultrasonic sensors have a cone of detection, the angle of this cone varies with distance, Figure 2 show this relation. The ability of a sensor to

detect an object also depends on the objects orientation to the sensor. If an object doesn't present a flat surface to the sensor then it is possible the sound wave will bounce off the object in a way that it does not return to the sensor.

Figure 2



2. HC-SR04 Specifications

The sensor chosen for the Firefighting Drone Project was the HC-SR04. This section contains the specifications and why they are important to the sensor module. The sensor modules requirements are as follows.

- Cost
- Weight
- Community of hobbyists and support
- Accuracy of object detection
- Probability of working in a smoky environment
- Ease of use

The HC-SR04 Specifications are listed below. These specifications are from the Cytron Technologies HC-SR04 User's Manual (source 1).

- Power Supply: +5V DC
- Quiescent Current: <2mA
- Working current: 15mA
- Effectual Angle: <15º
- Ranging Distance: 2-400 cm
- Resolution: 0.3 cm
- Measuring Angle: 30º
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm
- Weight: approx. 10 g

The HC-SR04's best selling point is its price; it can be purchased at around $2 per unit.

3. Timing Chart and Pin Explanations

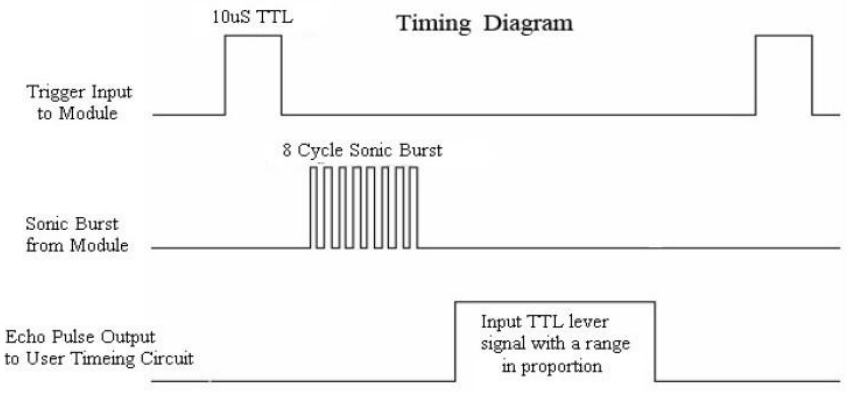
The HC-SR04 has four pins, VCC, GND, TRIG and ECHO; these pins all have different functions. The VCC and GND pins are the simplest -- they power the HC-SR04. These pins need to be attached to a +5 volt source and ground respectively. There is a single control pin: the TRIG pin. The TRIG pin is responsible for sending the ultrasonic burst. This pin should be set to HIGH for 10 μs, at which point the HC-SR04 will send out an eight cycle sonic burst at 40 kHZ. After a sonic burst has been sent the ECHO pin will go HIGH. The ECHO pin is the data pin -- it is used in taking distance measurements. After an ultrasonic burst is sent the pin will go HIGH, it will stay high until an ultrasonic burst is detected back, at which point it will go LOW.

Taking Distance Measurements

The HC-SR04 can be triggered to send out an ultrasonic burst by setting the TRIG pin to HIGH. Once the burst is sent the ECHO pin will automatically go HIGH. This pin will remain HIGH until the the burst hits the sensor again. You can calculate the distance to the object by keeping track of how long the ECHO pin stays HIGH. The time ECHO stays HIGH is the time the burst spent traveling. Using this measurement in equation 1 along with the speed of sound will yield the distance travelled. A summary of this is listed below, along with a visual representation in Figure 2.

1. Set TRIG to HIGH
2. Set a timer when ECHO goes to HIGH
3. Keep the timer running until ECHO goes to LOW
4. Save that time
5. Use equation 1 to determine the distance travelled

Figure 3
Source 2



Source 2

To interpret the time reading into a distance you need to change equation 1. The clock on the device you are using will probably count in microseconds or smaller. To use equation 1 the speed of sound needs to determined,which is 343 meters per second at standard temperature and pressure. To convert this into more useful form use equation 2 to change from meters per second to microseconds per centimeter. Then equation 3 can be used to easily compute the distance in centimeters.

Equation 2. $Distance = \frac{Speed}{170.15\ m} \times \frac{Meters}{100\ cm} \times \frac{1e6\ \mu S}{170.15\ m} \times \frac{58.772\ \mu S}{cm}$

Equation 3. $Distance = \frac{time}{58} = \frac{\mu s}{\mu s/cm} = cm$

4. Wiring the HC-SR04 to a Microcontroller

This section only covers the hardware side. For information on how to integrate the software side, look at one of the links below or look into the specific microcontroller you are using.

The HC-SR04 has 4 pins: VCC, GND, TRIG and ECHO.

1. VCC is a 5v power supply. This should come from the microcontroller
2. GND is a ground pin. Attach to ground on the microcontroller.
3. TRIG should be attached to a GPIO pin that can be set to HIGH
4. ECHO is a little more difficult. The HC-SR04 outputs 5v, which could destroy many microcontroller GPIO pins (the maximum allowed voltage varies). In order to step down the voltage use a single resistor or a voltage divider circuit. Once again this depends on the specific microcontroller you are using, you will need to find out its GPIO maximum voltage and make sure you are below that.

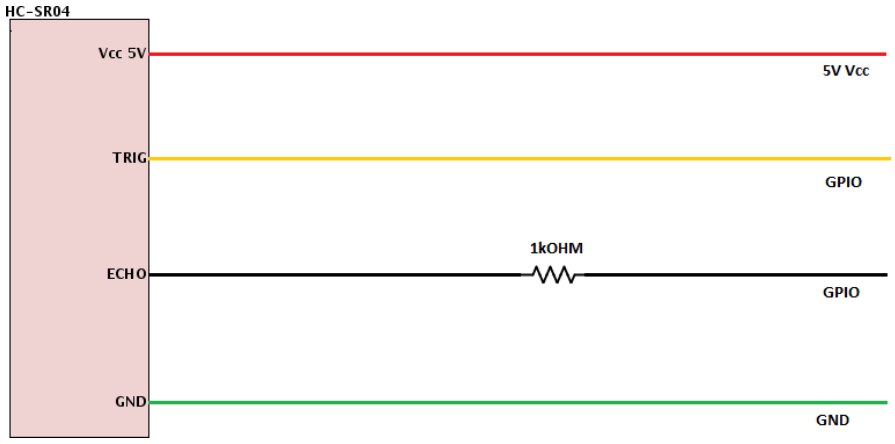
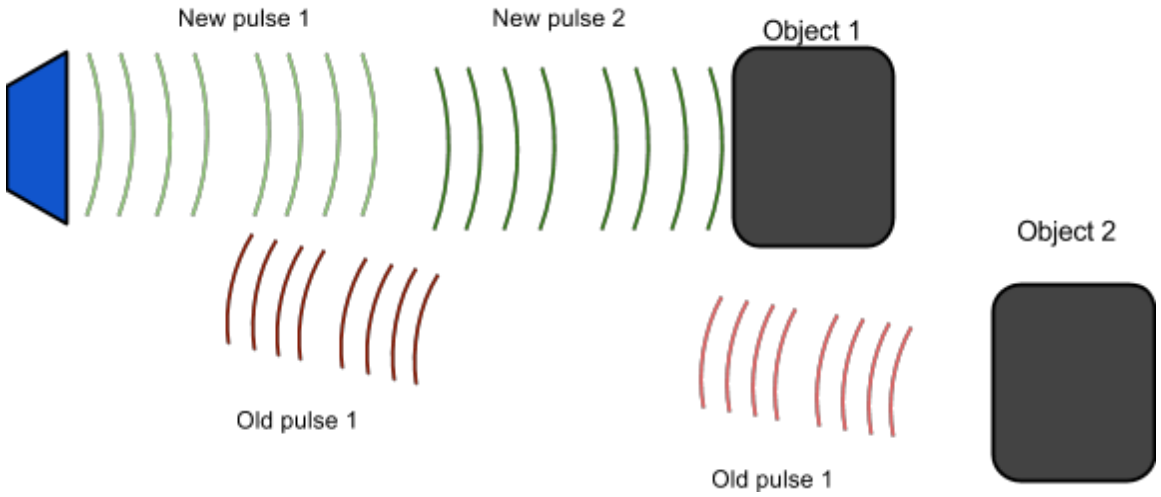


Figure 4

5. Errors and Bad Readings

Ultrasonic sensors are great sensors -- they work well for many applications where other types of sensors fall short. Unfortunately, they do have weaknesses. These weaknesses can be mitigated and worked around, but first they must be understood. The

first weakness is that they use sound. There is a limit to how fast ultrasonic sensors can get distance measurements. The longer the distance, the slower they are at reporting the distance. The second weakness comes from the way sound bounces off of objects. In enclosed spaces it is possible, if not probable that there will be unintended echos. The echos can very easily cause false short readings. In Figure 2 a pulse was sent out. It bounced off of object 1 and returned to the sensor. The distance was recorded and then a new pulse was sent. There was another object farther away, so that when the new pulse reaches object 1, the first signal will reach the sensor. This will cause the sensor to think that there is an object closer than is actually true. The old pulse is smaller than the new pulse because it has grown weaker. The longer the pulse exists the weaker it grows until it is negligible. If multiple sensors are being used, the number of echos will increase along with the number of errors. There are two main ways to reduce the number of errors. The first is to provide shielding around the sensor. This prevents echos coming in from angle outside what the sensor should actually pick up. The second is to reduce the frequency at which pulses are sent out. This gives more time for the echos to dissipate.



Works Cited

Source 1.
"HC-SR04 User's_Manual." *docs.google.* Cytron Technologies, May 2013 Web. 5 Dec. 2009.
<https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit>

Source 2.
"Attiny2313 Ultrasonic distance (HR-SR04) example." *CircuitDB.* n.a. 7 Sept. 2014 Web. 5 Dec. 2014. <http://www.circuitdb.com/?p=1162>

Links

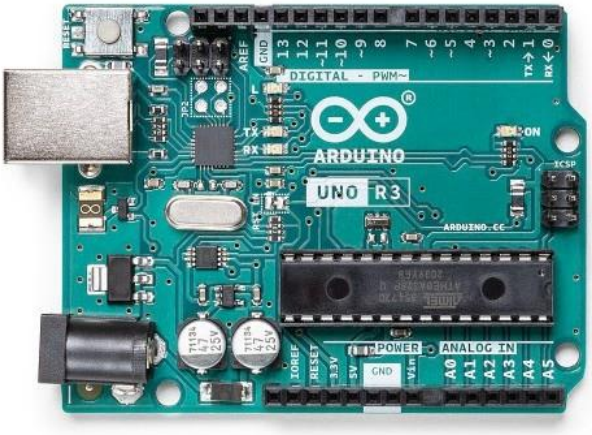
These are not formatted; you will need to copy and paste them into your web browser.

Want to learn about Ultrasonic Sensors in general?
http://www.sensorsmag.com/sensors/acoustic-ultrasound/choosing-ultrasonic-sensor-proximity-or-distance-measurement-825

All about the HC-SR04

- http://www.circuitdb.com/?p=1162
- http://www.micropik.com/PDF/HCSR04.pdf
- http://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/
- http://www.ezdenki.com/ultrasonic.php
  (^fantastic tutorial, explains a lot of stuff)
- http://www.elecrow.com/hcsr04-ultrasonic-ranging-sensor-p-316.html
  (^ this one has some cool charts)

User Manual
SKU: A000066



Description

The Arduino® UNO R3 is the perfect board to get familiar with electronics and coding. This versatile development board is equipped with the well-known ATmega328P and the ATMega 16U2 Processor.

This board will give you a great first experience within the world of Arduino.

Target areas:

Maker, introduction, industries

Features

- **ATMega328P** Processor
  - **Memory**
    - AVR CPU at up to 16 MHz
    - 32 kB Flash
    - 2 kB SRAM
    - 1 kB EEPROM
  - **Security**
    - Power On Reset (POR)
    - Brown Out Detection (BOD)
  - **Peripherals**
    - 2x 8-bit Timer/Counter with a dedicated period register and compare channels
    - 1x 16-bit Timer/Counter with a dedicated period register, input capture and compare channels
    - 1x USART with fractional baud rate generator and start-of-frame detection
    - 1x controller/peripheral Serial Peripheral Interface (SPI)
    - 1x Dual mode controller/peripheral I2C
    - 1x Analog Comparator (AC) with a scalable reference input
    - Watchdog Timer with separate on-chip oscillator
    - Six PWM channels
    - Interrupt and wake-up on pin change
  - **ATMega16U2 Processor**
    - 8-bit AVR® RISC-based microcontroller
  - **Memory**
    - 16 kB ISP Flash
    - 512B EEPROM
    - 512B SRAM
    - debugWIRE interface for on-chip debugging and programming
  - **Power**
    - 2.7-5.5 volts

CONTENTS

1 The Board

1.1 Application Examples

The UNO board is the flagship product of Arduino. Regardless if you are new to the world of electronics or will use the UNO R3 as a tool for education purposes or industry-related tasks, the UNO R3 is likely to meet your needs.

First entry to electronics: If this is your first project within coding and electronics, get started with our most used and documented board; UNO. It is equipped with the well-known ATmega328P processor, 14 digital input/output pins, 6 analog inputs, USB connections, ICSP header and reset button. This board includes everything you will need for a great first experience with Arduino.

Industry-standard development board: Using the UNO R3 board in industries, there are a range of companies using the UNO R3 board as the brain for their PLC's.

Education purposes: Although the UNO R3 board has been with us for about ten years, it is still widely used for various education purposes and scientific projects. The board's high standard and top quality performance makes it a great resource to capture real time from sensors and to trigger complex laboratory equipment to mention a few examples.

1.2 Related Products

- Arduino Starter Kit
- Arduino UNO R4 Minima
- Arduino UNO R4 WiFi
- Tinkerkit Braccio Robot

2 Ratings

2.1 Recommended Operating Conditions

Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (-40 °F)	85 °C (185 °F)

NOTE: In extreme temperatures, EEPROM, voltage regulator, and the crystal oscillator, might not work as expected.

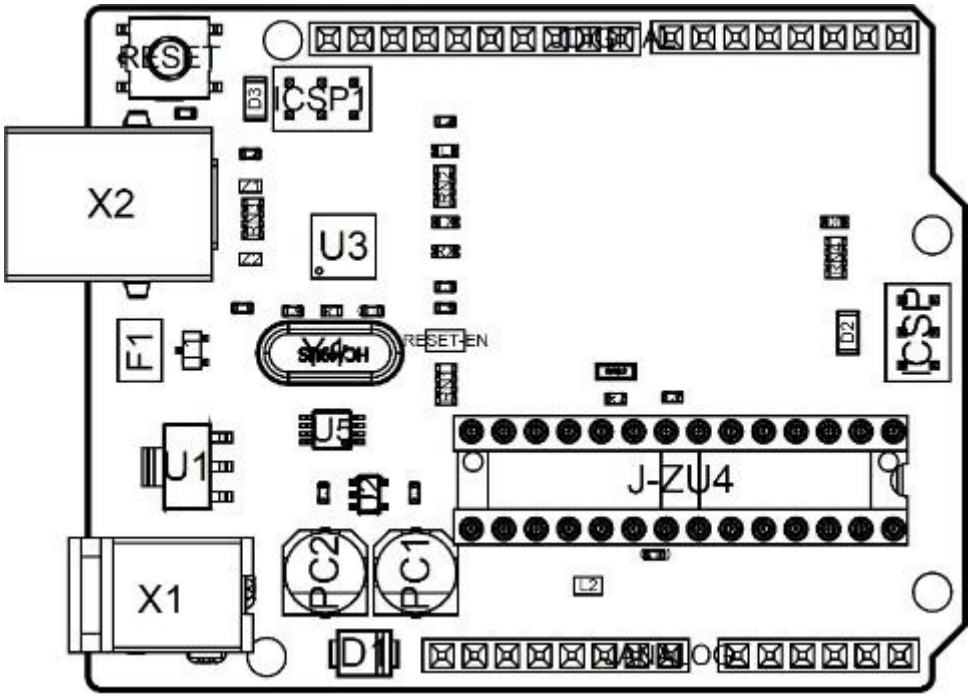
2.2 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
VINMax	Maximum input voltage from VIN pad	6	-	20	V
VUSBMax	Maximum input voltage from USB connector		-	5.5	V
PMax	Maximum Power Consumption	-	-	xx	mA

3 Functional Overview

3.1 Board Topology

Top view



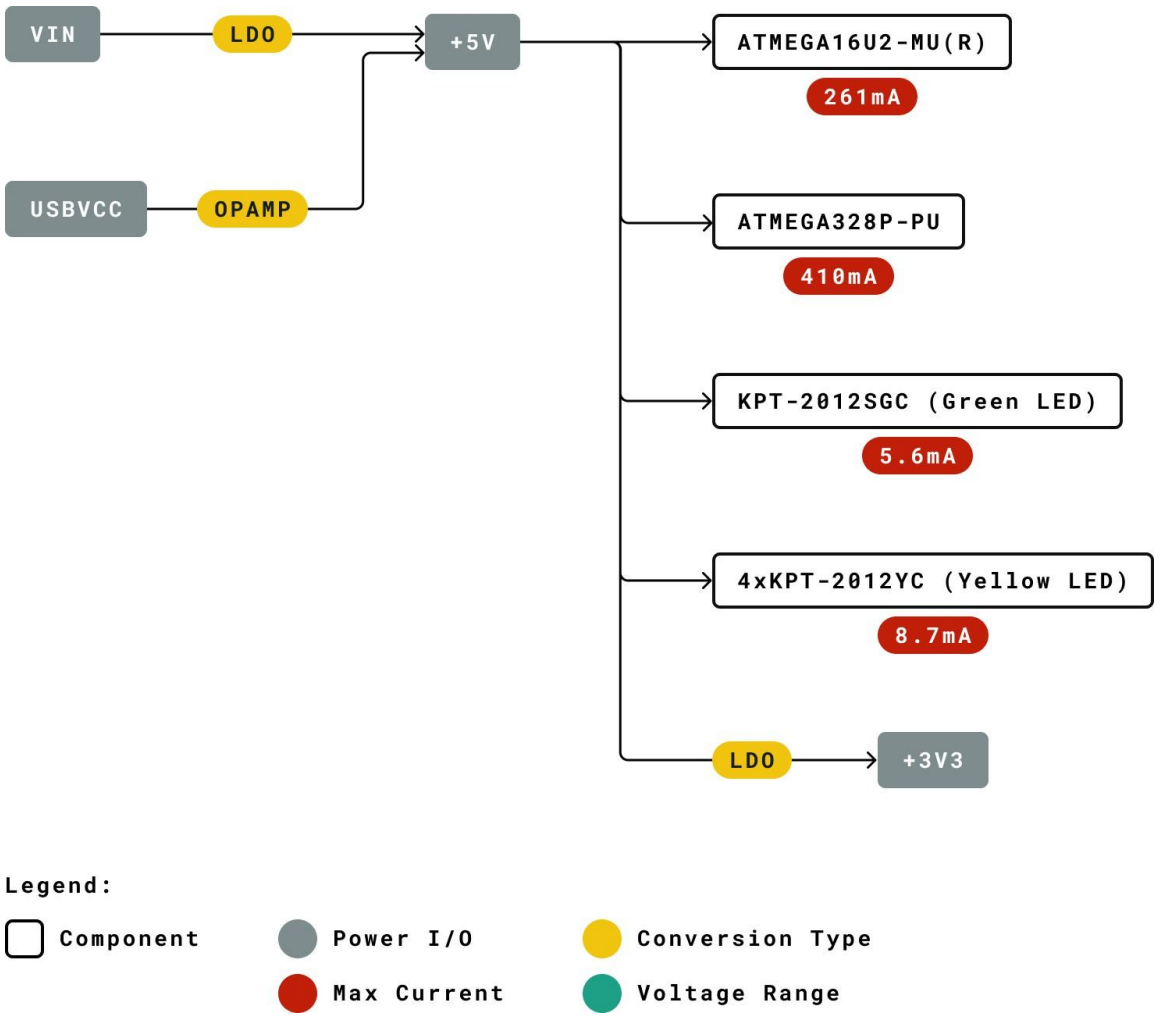
Board topology

Ref.	Description	Ref.	Description
X1	Power jack 2.1x5.5mm	U1	SPX1117M3-L-5 Regulator
X2	USB B Connector	U3	ATMEGA16U2 Module
PC1	EEE-1EA470WP 25V SMD Capacitor	U5	LMV358LIST-A.9 IC
PC2	EEE-1EA470WP 25V SMD Capacitor	F1	Chip Capacitor, High Density
D1	CGRA4007-G Rectifier	ICSP	Pin header connector (through hole 6)
J-ZU4	ATMEGA328P Module	ICSP1	Pin header connector (through hole 6)
Y1	ECS-160-20-4X-DU Oscillator		

3.2 Processor

The Main Processor is a ATmega328P running at up to 20 MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the USB Bridge coprocessor.

3.3 Power Tree



Power tree

4 Board Operation

4.1 Getting Started - IDE

If you want to program your UNO R3 while offline you need to install the Arduino Desktop IDE [1] To connect the UNO R3 to your computer, you'll need a USB-B cable. This also provides power to the board, as indicated by the LED.

4.2 Getting Started - Arduino Cloud Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino Cloud Editor [2], by just installing a simple plugin.

The Arduino Cloud Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

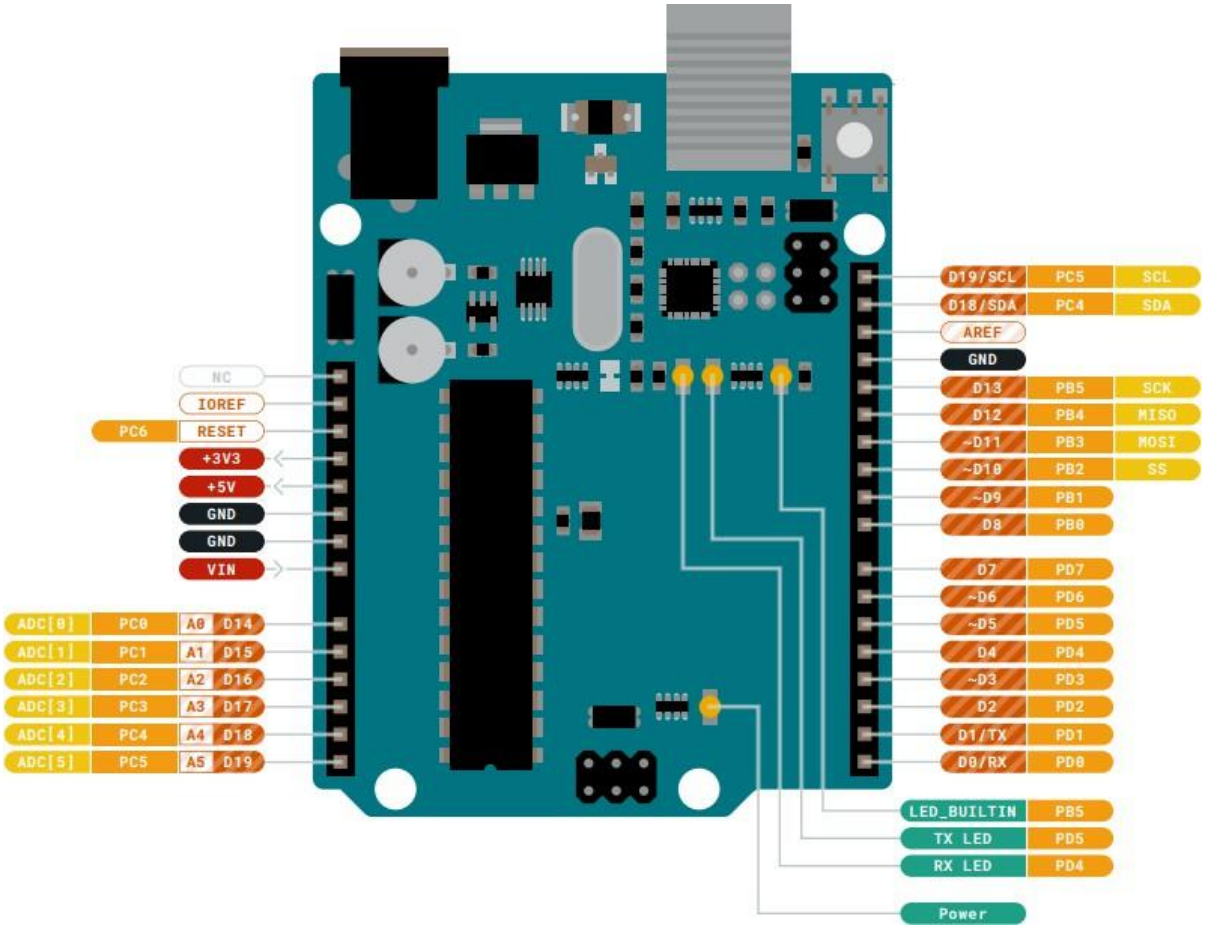
4.3 Sample Sketches

Sample sketches for the UNO R3 can be found either in the "Examples" menu in the Arduino IDE or in the "Documentation" section of the Arduino website [4].

4.4 Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on Arduino Project Hub [5], the Arduino Library Reference [6] and the online Arduino store [7] where you will be able to complement your board with sensors, actuators and more.

5 Connector Pinouts



Pinout

5.1 JANALOG

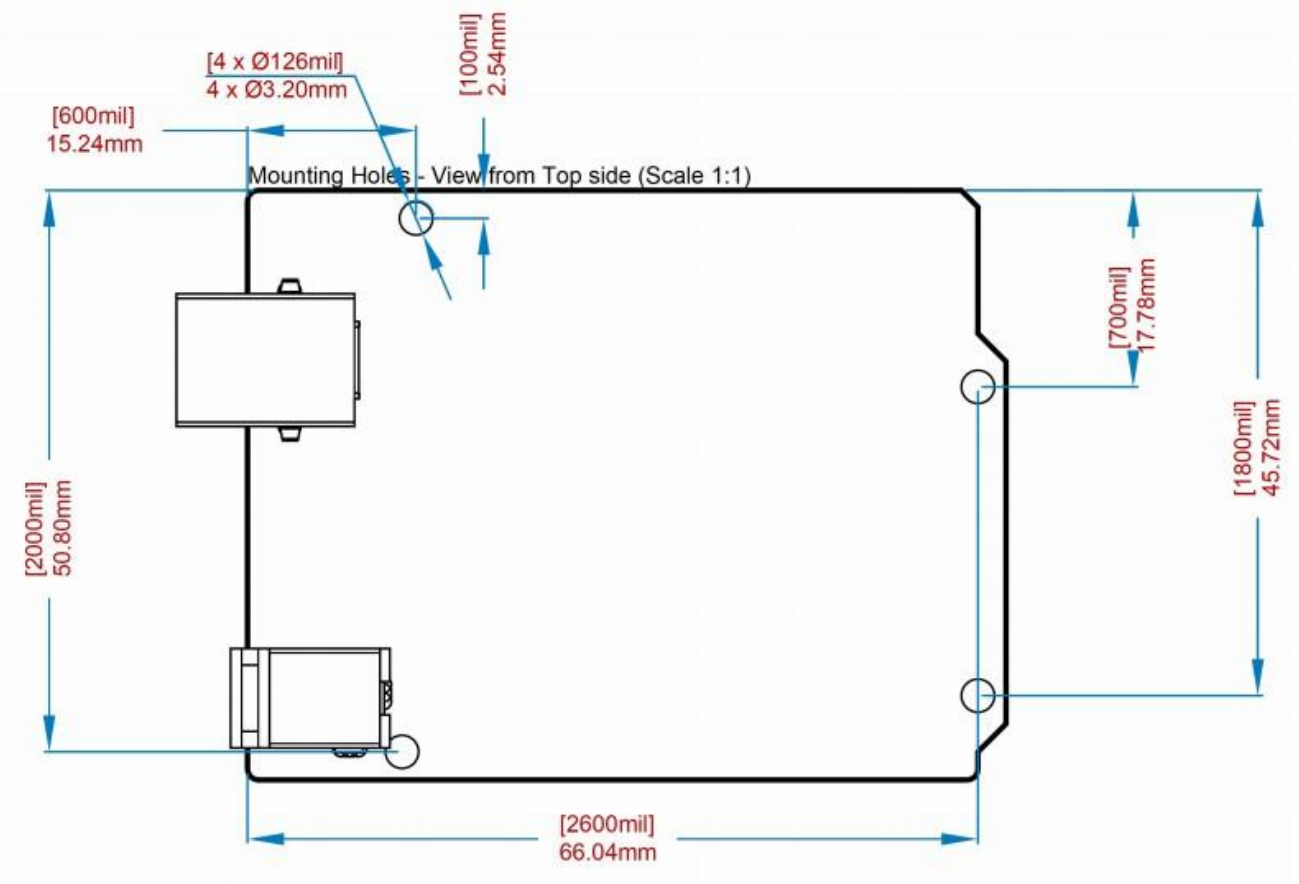
Pin	Function	Type	Description
1	NC	NC	Not connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog/GPIO	Analog input 0 /GPIO
10	A1	Analog/GPIO	Analog input 1 /GPIO
11	A2	Analog/GPIO	Analog input 2 /GPIO
12	A3	Analog/GPIO	Analog input 3 /GPIO
13	A4/SDA	Analog input/I2C	Analog input 4/I2C Data line
14	A5/SCL	Analog input/I2C	Analog input 5/I2C Clock line

5.2 JDIGITAL

Pin	Function	Type	Description
1	D0	Digital/GPIO	Digital pin 0/GPIO
2	D1	Digital/GPIO	Digital pin 1/GPIO
3	D2	Digital/GPIO	Digital pin 2/GPIO
4	D3	Digital/GPIO	Digital pin 3/GPIO
5	D4	Digital/GPIO	Digital pin 4/GPIO
6	D5	Digital/GPIO	Digital pin 5/GPIO
7	D6	Digital/GPIO	Digital pin 6/GPIO
8	D7	Digital/GPIO	Digital pin 7/GPIO
9	D8	Digital/GPIO	Digital pin 8/GPIO
10	D9	Digital/GPIO	Digital pin 9/GPIO
11	SS	Digital	SPI Chip Select
12	MOSI	Digital	SPI1 Main Out Secondary In
13	MISO	Digital	SPI Main In Secondary Out
14	SCK	Digital	SPI serial clock output
15	GND	Power	Ground
16	AREF	Digital	Analog reference voltage
17	A4/SD4	Digital	Analog input 4/I2C Data line (duplicated)
18	A5/SD5	Digital	Analog input 5/I2C Clock line (duplicated)

5.3 Mechanical Information

5.4 Board Outline & Mounting Holes



Board outline

6 Certifications

6.1 Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

ROHS 2 Directive 2011/65/EU	
Conforms to:	EN50581:2012
Directive 2014/35/EU. (LVD)	
Conforms to:	EN 60950-1:2006/A11:2009/A1:2010/A12:2011/AC:2011
Directive 2004/40/EC & 2008/46/EC & 2013/35/EU, EMF	
Conforms to:	EN 62311:2008

6.2 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	Maximum limit (ppm)
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl} phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions: No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

6.3 Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regards to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder, or as a component in metal alloys. As part of our reasonable due diligence Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

7 FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with minimum distance 20cm between the radiator & your body.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of the device.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes :

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

IC SAR Warning:

English This equipment should be installed and operated with minimum distance 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85°C and shouldn't be lower than -40°C.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

8 Company Information

Company name	**Arduino S.r.l**
Company Address	Via Andrea Appiani 25 20900 MONZA Italy

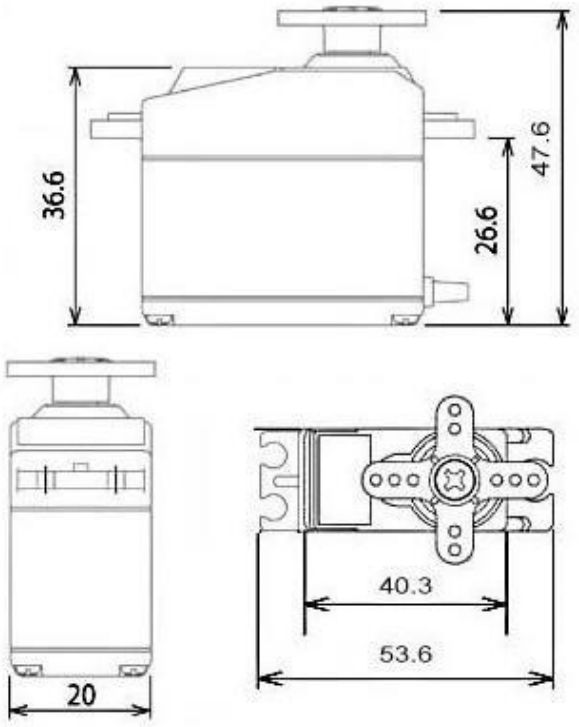
9 Reference Documentation

Reference	**Link**
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino Cloud Editor	https://create.arduino.cc/editor
Arduino Cloud Editor - Getting Started	https://docs.arduino.cc/arduino-cloud/guides/editor/
Arduino Website	https://www.arduino.cc/
Arduino Project Hub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://www.arduino.cc/reference/en/
Arduino Store	https://store.arduino.cc/

10 Revision History

Date	**Revision**	**Changes**
25/04/2024	3	Updated link to new Cloud Editor
26/07/2023	2	General Update
06/2021	1	Datasheet release

MG996R High Torque
Metal Gear Dual Ball Bearing Servo



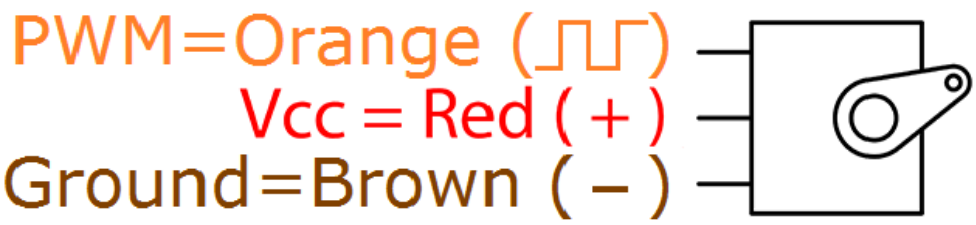
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

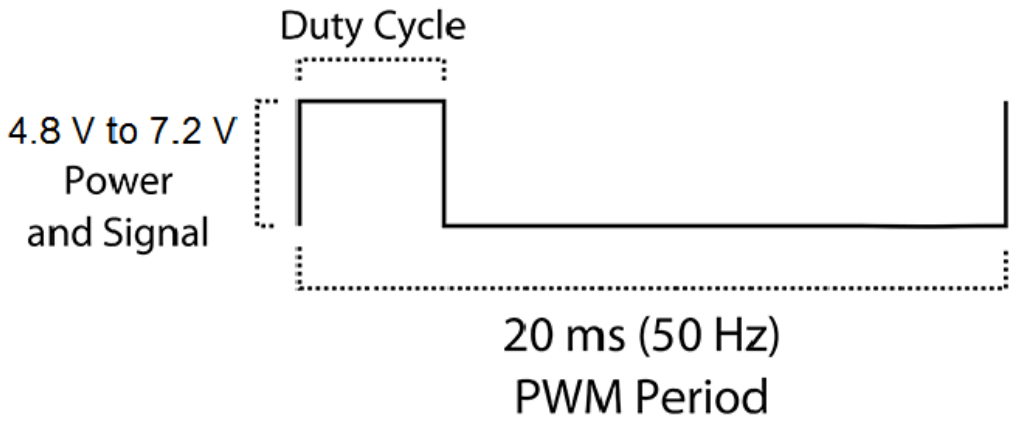
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 55 ºC





Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

4x4 Matrix Membrane Keypad (#27899)

This 16-button keypad provides a useful human interface component for microcontroller projects. Convenient adhesive backing provides a simple way to mount the keypad in a variety of applications.

Features

- Ultra-thin design
- Adhesive backing
- Excellent price/performance ratio
- Easy interface to any microcontroller
- Example programs provided for the BASIC Stamp 2 and Propeller P8X32A microcontrollers

Key Specifications

- Maximum Rating: 24 VDC, 30 mA
- Interface: 8-pin access to 4x4 matrix
- Operating temperature: 32 to 122 °F (0 to 50°C)
- Dimensions:
  Keypad, 2.7 x 3.0 in (6.9 x 7.6 cm)
  Cable: 0.78 x 3.5 in (2.0 x 8.8 cm)

Application Ideas

- Security systems
- Menu selection
- Data entry for embedded systems

How it Works

Matrix keypads use a combination of four rows and four columns to provide button states to the host device, typically a microcontroller. Underneath each key is a pushbutton, with one end connected to one row, and the other end connected to one column. These connections are shown in Figure 1.

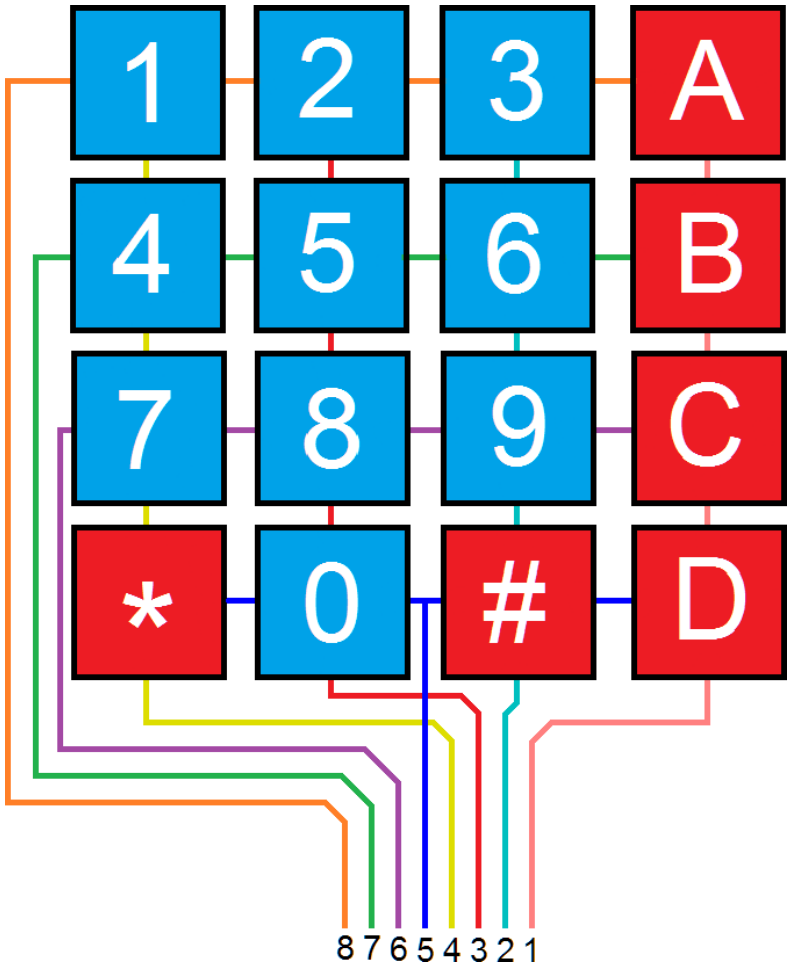


Figure 1: Matrix Keypad Connections

In order for the microcontroller to determine which button is pressed, it first needs to pull each of the four columns (pins 1-4) either low or high one at a time, and then poll the states of the four rows (pins 5-8). Depending on the states of the columns, the microcontroller can tell which button is pressed.

For example, say your program pulls all four columns low and then pulls the first row high. It then reads the input states of each column, and reads pin 1 high. This means that a contact has been made between column 4 and row 1, so button 'A' has been pressed.

Connection Diagrams

Figure 2

For use with the BASIC Stamp example program listed below.

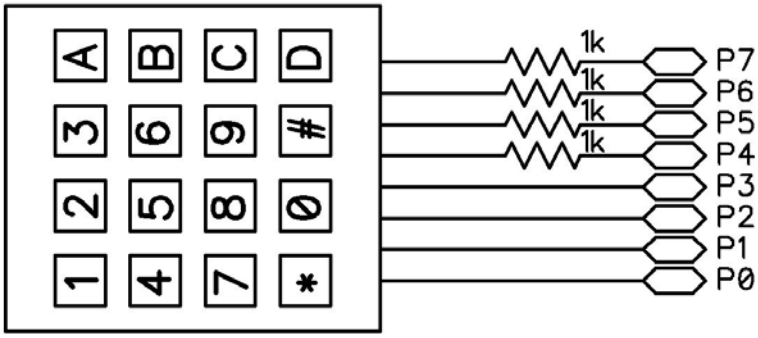
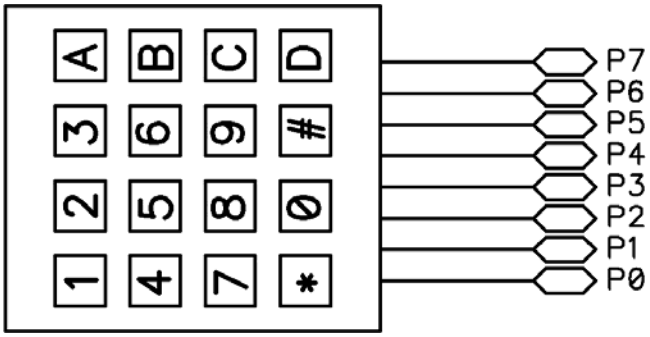


Figure 3

For use with the Propeller P8X32A example program listed below.



BASIC Stamp® Example Code

The example code below displays the button states of the 4x4 Matrix Membrane Keypad. It uses the Debug Terminal, which is built into the BASIC Stamp Editor software. The software is a free download from www.parallax.com/basicstampsoftware.

```
' 4x4MatrixKeypad_Demo.bs2
' Display buttons pressed on the 4x4 Matrix Membrane Keypad
' Author: Parallax HK Engineering

' {$STAMP BS2}
' {$PBASIC 2.5}

row         VAR  Nib                    ' Variable space for row counting
column      VAR  Nib                    ' Variable space for column counting
keypad     VAR    Word                  ' Variable space to store keypad output
keypadOld   VAR   Word                  ' Variable space to store old keypad output
temp       VAR  Nib                     ' Variable space for polling column states

DEBUG CLS                               ' Clear Debug Terminal
GOSUB Update                            ' Display keypad graphic

DO
  GOSUB ReadKeypad                      ' Read keypad button states
  DEBUG HOME, BIN16 keypad, CR, CR,     ' Display 16-bit keypad value
              BIN4 keypad >> 12,CR,     ' Display 1st row 4-bit keypad value
              BIN4 keypad >> 8, CR,     ' Display 2nd row 4-bit keypad value
              BIN4 keypad >> 4, CR,     ' Display 3rd row 4-bit keypad value
              BIN4 keypad               ' Display 4th row 4-bit keypad value
```

```
  IF keypad <> keypadOld THEN               ' If different button is pressed,
    GOSUB Update                            ' update the keypad graphic to clear
  ENDIF                                     ' old display

  IF keypad THEN                            ' Display button pressed in graphic
    GOSUB display
  ENDIF

  keypadOld = keypad                        ' Store keypad value in variable keypadOld
LOOP

' -----[ Subroutine - ReadKeypad ]-------------------------------------------------
' Read keypad button states
ReadKeypad:
  keypad = 0
  OUTL   = %00000000                        ' Initialize IO
  DIRL   = %00000000

  FOR row = 0 TO 3
    DIRB = %1111                            ' Set columns (P7-P4) as outputs
    OUTB = %0000                            ' Pull columns low (act as pull down)
    OUTA = 1 << row                         ' Set rows high one by one
    DIRA = 1 << row

    temp = 0                                ' Reset temp variable to 0
    FOR column = 0 TO 3
      INPUT (column + 4)                    ' Set columns as inputs
      temp = temp | (INB & (1 << column))   ' Poll column state and store in temp
    NEXT

    keypad = keypad << 4 | (Temp REV 4)     ' Store keypad value
  NEXT
RETURN

' -----[ Subroutine - Update ]-----------------------------------------------------
' Graphical depiction of keypad
Update:
  DEBUG CRSRXY,0,7,
    "+---+---+---+---+",CR,
    "|   |   |   |   |",CR,
    "+---+---+---+---+",CR,
    "|   |   |   |   |",CR,
    "+---+---+---+---+",CR,
    "|   |   |   |   |",CR,
    "+---+---+---+---+",CR,
    "|   |   |   |   |",CR,
    "+---+---+---+---+"
RETURN

' -----[ Subroutine - Display ]---------------------------------------------------
' Display button pressed in keypad graphic
Display:
  IF KeyPad.BIT15 THEN  DEBUG CRSRXY, 02,08,"1"
  IF Keypad.BIT14 THEN  DEBUG CRSRXY, 06,08,"2"
  IF KeyPad.BIT13 THEN  DEBUG CRSRXY, 10,08,"3"
  IF Keypad.BIT12 THEN  DEBUG CRSRXY, 14,08,"A"
  IF KeyPad.BIT11 THEN  DEBUG CRSRXY, 02,10,"4"
  IF Keypad.BIT10 THEN  DEBUG CRSRXY, 06,10,"5"
  IF KeyPad.BIT9  THEN  DEBUG CRSRXY, 10,10,"6"
  IF Keypad.BIT8  THEN  DEBUG CRSRXY, 14,10,"B"
  IF KeyPad.BIT7  THEN  DEBUG CRSRXY, 02,12,"7"
  IF Keypad.BIT6  THEN  DEBUG CRSRXY, 06,12,"8"
  IF KeyPad.BIT5  THEN  DEBUG CRSRXY, 10,12,"9"
```

```
  IF Keypad.BIT4  THEN  DEBUG CRSRXY, 14,12,"C"
  IF KeyPad.BIT3  THEN  DEBUG CRSRXY, 02,14,"*"
  IF Keypad.BIT2  THEN  DEBUG CRSRXY, 06,14,"0"
  IF KeyPad.BIT1  THEN  DEBUG CRSRXY, 10,14,"#"
  IF Keypad.BIT0  THEN  DEBUG CRSRXY, 14,14,"D"
RETURN
```

Propeller™ P8X32A Example Code

The example code below displays the button states of the 4x4 Matrix Membrane Keypad, and is a modified version of the 4x4 Keypad Reader DEMO object by Beau Schwabe.

Note: This application uses the 4x4 Keypad Reader.spin object. It also uses the Parallax Serial Terminal to display the device output. Both objects and the Parallax Serial Terminal itself are included with the with the Propeller Tool v1.2.7 or higher, which is available from the Downloads link at www.parallax.com/Propeller.

```
{{ 4x4 Keypad Reader PST.spin
Returns the entire 4x4 keypad matrix into a single WORD variable indicating which buttons are
pressed. }}

CON

  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

OBJ
  text : "Parallax Serial Terminal"
  KP   : "4x4 Keypad Reader"

VAR
  word  keypad

PUB start
  'start term
  text.start(115200)
  text.str(string(13,"4x4 Keypad Demo..."))
  text.position(1, 7)
  text.str(string(13,"RAW keypad value 'word'"))

  text.position(1, 13)
  text.str(string(13,"Note: Try pressing multiple keys"))


  repeat
    keypad := KP.ReadKeyPad      '<-- One line command to read the 4x4 keypad
    text.position(5, 2)
    text.bin(keypad>>0, 4)       'Display 1st ROW
    text.position(5,3)
    text.bin(keypad>>4, 4)       'Display 2nd ROW
    text.position(5, 4)
    text.bin(keypad>>8, 4)       'Display 3rd ROW
    text.position(5, 5)
    text.bin(keypad>>12, 4)      'Display 4th ROW
    text.position(5, 9)
    text.bin(keypad, 16)         'Display RAW keypad value
```

Revision History

v1.0: original document
v1.1: Updated Figure 1 on page 2
v1.2: Updated Figure 1 on page 2 (again); updated BS2 comment