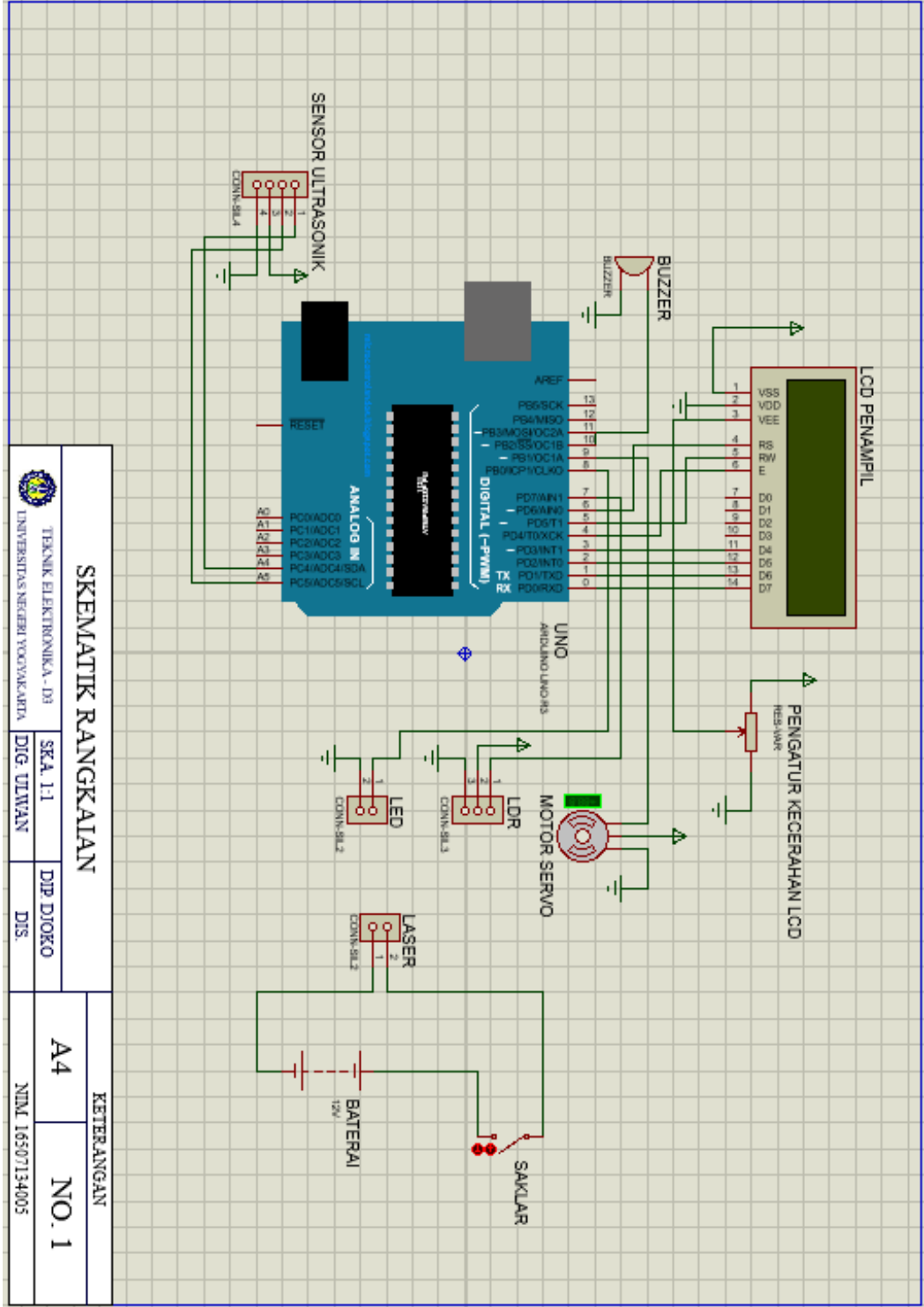


LAMPIRAN

Lampiran 1. Skema Elektronik Rangkaian



Lampiran 2. Foto Alat





Lampiran 3. Foto Papan Pantul



Lampiran 4. *Listing Program*

```
#include <LiquidCrystal.h>
#include <Servo.h>
#define SCL_PIN 5
#define SCL_PORT PORTC
#define SDA_PIN 4
#define SDA_PORT PORTC
#define I2C_TIMEOUT 100
#include <SoftI2CMaster.h>

Servo servoBendera;
const int rs = A3, en = A2, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

int range;
int sensorLDR;
int i;
int pinLDR = 8;
int pinLED = 7;
int pinBuzzer = 11;
long period1 = 50;
long period2 = 50;
long period3 = 50;
long period4 = 50;
long period5 = 50;
long period6 = 50;
long time_1 = 0;
long time_2 = 0;
long time_3 = 0;
long time_4 = 0;
long time_5 = 0;
long time_6 = 0;

void setup(){
  Serial.begin(9600);
  pinMode(pinLDR,INPUT);
  pinMode(pinBuzzer,OUTPUT);
  pinMode(pinLED,OUTPUT);
  i2c_init();
```

```

address_polling_example();
default_address_change_example();
lcd.begin(16, 2);
lcd.setCursor(0,0);
lcd.print("Loading...");
lcd.setCursor(7,1);
lcd.print("System...");
servoBendera.attach(6);
servoBendera.write(90);
delay(5000);
lcd.clear();
}

void loop(){
  if(millis() - time_1 > period1){
    read_the_sensor_example();
    time_1 = millis();
  }
}

void blinkLB(){
  for(i = 1; i <= 25; i++){
    digitalWrite(pinLED,HIGH);
    digitalWrite(pinBuzzer,HIGH);
    delay(100);
    digitalWrite(pinLED,LOW);
    digitalWrite(pinBuzzer,LOW);
    delay(100);
  }
}

void servoAktif(){
  servoBendera.write(180);
}

void servoTidakAktif(){
  servoBendera.write(90);
}

boolean start_sensor(byte bit8address){

```

```

    boolean errorlevel = 0;
    bit8address = bit8address & B11111110;
    errorlevel = !i2c_start(bit8address) | errorlevel;
    errorlevel = !i2c_write(81) | errorlevel;
    i2c_stop();
    return errorlevel;
}

int read_sensor(byte bit8address){
    boolean errorlevel = 0;
    int range = 0;
    byte range_highbyte = 0;
    byte range_lowbyte = 0;
    bit8address = bit8address | B00000001;
    errorlevel = !i2c_start(bit8address) | errorlevel;
    range_highbyte = i2c_read(0);
    range_lowbyte = i2c_read(1);
    i2c_stop();
    range = (range_highbyte * 256) + range_lowbyte;
    if(errorlevel){
        return 0;
    }
    else{
        return range;
    }
}

boolean change_address(byte oldaddress,byte newaddress){
    boolean errorlevel = 0;
    oldaddress = oldaddress & B11111110;
    errorlevel = !i2c_start(oldaddress) | errorlevel;
    errorlevel = !i2c_write(170) | errorlevel;
    errorlevel = !i2c_write(165) | errorlevel;
    errorlevel = !i2c_write(newaddress) | errorlevel;
    i2c_stop();
    return errorlevel;
}

void read_the_sensor_example(){
    boolean error = 0;
    error = start_sensor(222);

```



```

if (!error){
  delay(100);
  range = read_sensor(222);
  Serial.print("R:");Serial.println(range);
  if(millis() - time_2 > period2){
    lcd.setCursor(0,0);
    lcd.print("Jarak : ");
    lcd.print(range);
    lcd.print(" cm ");
    time_2 = millis();
  }
  if(millis() - time_3 > period3){
    sensorLDR = digitalRead(pinLDR);
    if(millis() - time_4 > period4){
      if(sensorLDR==LOW){
        digitalWrite(pinLED,LOW);
        digitalWrite(pinBuzzer,LOW);
        lcd.setCursor(0,1);
        lcd.print("STATUS SAH   ");
        servoTidakAktif();
      }
      time_4 = millis();
    }
    if(millis() - time_5 > period5){
      if(sensorLDR==HIGH){
        lcd.setCursor(0,0);
        lcd.print("Jarak : ");
        lcd.print("0 cm ");
        lcd.setCursor(0,1);
        lcd.print("STATUS TIDAK SAH !");
        servoAktif();
        if(millis() - time_6 > period6){
          blinkLB();
          time_6 = millis();
        }
      }
      time_3 = millis();
    }
  }
}

```

```

}
}
void address_polling_example(){
    boolean error = 0;
    int range = 0;
    Serial.println("Polling addresses...");
    for (byte i=2; i!=0; i+=2){
        error = 0;
        error = start_sensor(i);
        if (!error){
            delay(100);
            range = read_sensor(i);
            Serial.println(i);
            if (range != 0){
                Serial.print("Device found at:");Serial.print(i);Serial.print(" Reported value
of:");Serial.println(range);
            }
        }
        else{
            Serial.print("Couldn't start:");Serial.println(i);
        }
    }
    Serial.println("Address polling complete.");
}
void default_address_change_example(){
    boolean error = 0;
    int range;
    Serial.println("Take a reading at the default address");
    error = start_sensor(222);
    if (!error){
        delay(100);
        range = read_sensor(222);
        Serial.print("R:");Serial.println(range);
    }
    Serial.println("Change the sensor at the default address to 222");
    error = 0;
    error = change_address(224,222);
    delay(200);
    Serial.println("Take a reading at the new address");
}

```

```
error = 0;

error = start_sensor(222);
if (!error){
    delay(100);
    range = read_sensor(222);
    Serial.print("N:");Serial.println(range);
}
Serial.println("Change the sensor back to the default address");
error = 0;
error = change_address(222,224);
delay(200);
}
```

Lampiran 5. Daftar Komponen

1. Arduino UNO
2. Sensor Ultrasonik GY-US42V2 MB1242
3. LCD 16x2
4. Laser KY-008
5. Modul LDR
6. *Buzzer* 5v
7. LED 5mm
8. Motor *Servo* SG90
9. Baterai Kotak 9v
10. Sakelar
11. Boks Hitam 15x10x5 cm
12. Papan Pantul dari triplek 56x24 cm

Lampiran 6. IAAF Competition Rules 2018-2019 bagian Lompat Jauh

IAAF COMPETITION RULES 2018-2019

positioned $1.22\text{m} \pm 0.05\text{m}$ high and not more than 2m away from the runway.

12. The wind velocity shall be measured for a period of 5 seconds from the time an athlete passes a mark placed alongside the runway, for the Long Jump 40m from the take-off line and for the Triple Jump 35m. If an athlete runs less than 40m or 35m, as appropriate, the wind velocity shall be measured from the time he commences his run.

RULE 185

Long Jump

Competition

1. An athlete fails if:
 - (a) he while taking off, touches the ground (including any part of the plasticine board) beyond the take-off line with any part of his body, whether running up without jumping or in the act of jumping; or
 - (b) he takes off from outside either end of the board, whether beyond or before the extension of the take-off line; or
 - (c) he employs any form of somersaulting whilst running up or in the act of jumping; or
 - (d) after taking off, but before his first contact with the landing area, he touches the runway or the ground outside the runway or outside the landing area; or
 - (e) in the course of landing (including any overbalancing), he touches the border of, or the ground outside, the landing area closer to the take-off line than the nearest break made in the sand; or
 - (f) he leaves the landing area in any manner other than that described in Rule 185.2.
2. When leaving the landing area, an athlete's first contact by foot with its border or the ground outside shall be further from the take-off line than the nearest break in the sand (which may be any mark made on overbalancing completely inside the landing area or when walking back closer to the take-off line than the initial break on landing).

Note: This first contact is considered leaving.

3. An athlete shall not be regarded to have failed if:
 - (a) he runs outside the white lines marking the runway at any point; or

IAAF COMPETITION RULES 2018-2019

- (b) except as described in Rule 185.1(b), he takes off before reaching the board; or
- (c) under Rule 185.1(b) a part of his shoe / foot is touching the ground outside either end of the take-off board, before the take-off line; or
- (d) if in the course of landing, he touches, with any part of his body, or anything attached to it at that moment, the border of, or the ground outside the landing area, unless such contact contravenes Rule 185.1(d) or (e); or
- (e) he walks back through the landing area after having left the landing area in the manner described in Rule 185.2.

Take-off Line

- 4. The distance between the take-off line and the far end of the landing area shall be at least 10m.
- 5. The take-off line shall be placed between 1m and 3m from the nearer end of the landing area.

Team of Officials

For a Long Jump or Triple Jump event, it is recommended to allocate the available officials as follows:

- (1) The Chief Judge will watch over the whole of the event.
- (2) Judge checking whether the take-off has been made correctly and measuring the trial. He must be provided with two flags - white to indicate if the trial is valid and red if it is a failure. When the jump has been measured, it is advised that the judge stands in front of the take-off board, holding the red flag, while the landing area is levelled and, if relevant, the plasticine board is replaced. A cone may be used instead or in addition. (In some competitions this position is assumed by the Chief Judge of the event.)
- (3) Judge at the landing point determining the position of the nearest break in the landing area to the take-off line, to insert the spike/prism and then, if a tape is being used, hold the tape on the 0. When video measuring is being used, no judge will normally be required on site for this purpose. When an optic system of measuring is being used on site, two judges are needed at the landing point, one to plant the marker in the sand, the other one to read the result on the optic apparatus.
- (4) Judge - a recorder scoring the results sheet and calling each athlete (and the one who is to follow).
- (5) Judge in charge of the scoreboard (trial-number-result).

RULE 185

152

- (6) Judge in charge of the wind-gauge positioned at a point 20 metres from the take-off board.
- (7) one or more Judges or assistants in charge of levelling the landing area after each trial.
- (8) Judge or an assistant in charge of replacing the plasticine.
- (9) Judge in charge of the clock indicating to the athletes that they have a certain time to take their trial.
- (10) Judge in charge of athletes.

Note (i): This is the traditional setting-up of the officials. In major competitions, where a data system and electronic scoreboards are available, specialised personnel are certainly required. To be clear in these cases, the progress and scoring of a Field Event is followed by both the recorder and by the data system.

Note (ii): Officials and equipment must be placed in such a way as not to obstruct the athlete's way nor impede the view of the spectators.

Note (iii): A space must be reserved for a wind-sock to indicate the wind direction and strength.

RULE 186 **Triple Jump**

Rules 184 and 185 apply to Triple Jump with the following variations:

Competition

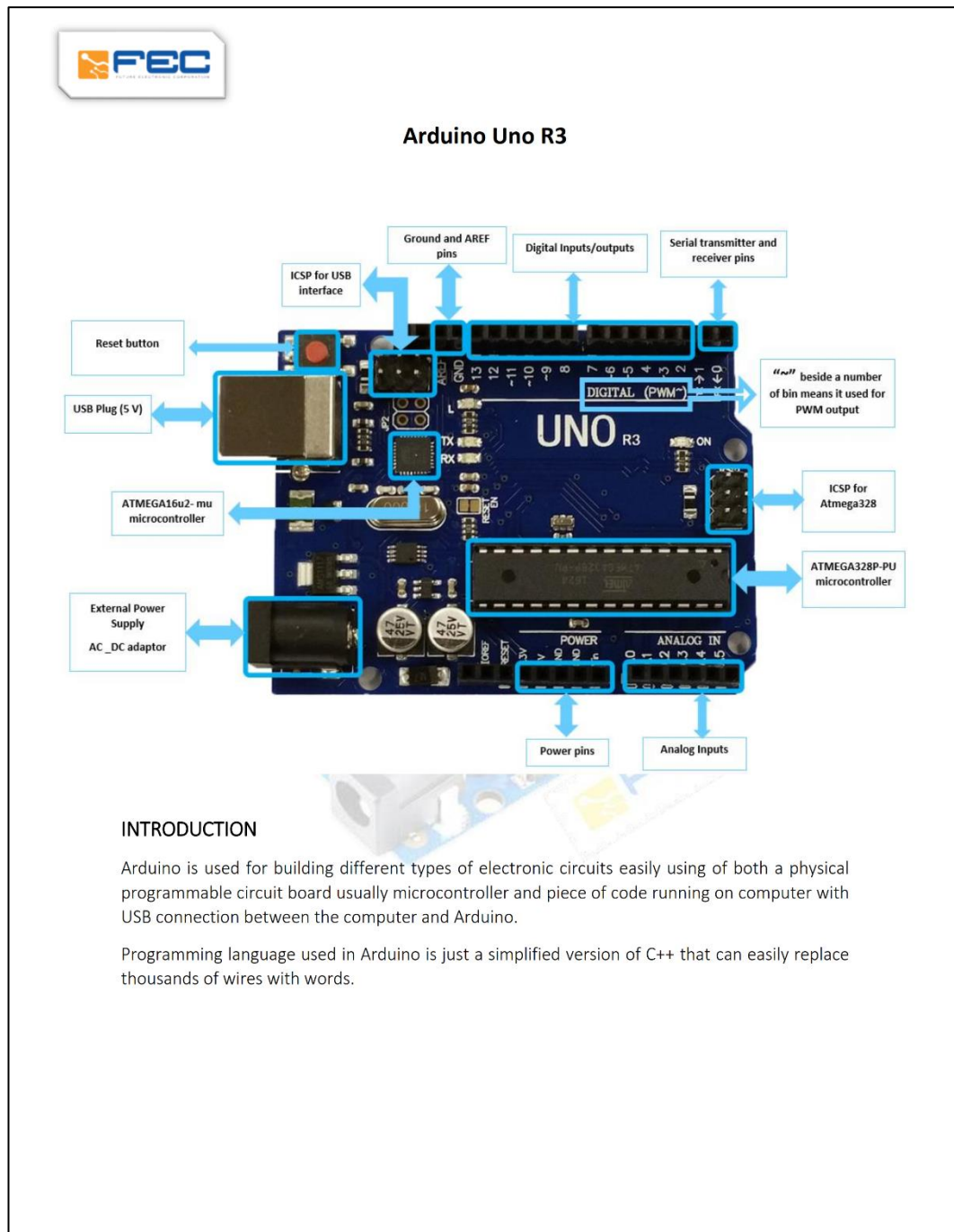
1. The Triple Jump shall consist of a hop, a step and a jump in that order.
2. The hop shall be made so that an athlete lands first on the same foot as that from which he has taken off; in the step he shall land on the other foot, from which, subsequently, the jump is performed.
It shall not be considered a failure if an athlete, while jumping, touches the ground with the "sleeping" leg.

Note: Rule 185.1(d) does not apply to the normal landings from the hop and step phases.

It should be noted that it is not a failure (for that reason alone) if the athlete:

- (a) touches the white lines or the ground outside between the take-off line and the landing area; or
- (b) if the athlete lands in the pit in the step phase through no fault of his own (i.e. if the Judge incorrectly indicated the take-off board) - in which such case the Referee would normally offer the athlete a substitute trial.

Lampiran 7. Datasheet Arduino UNO



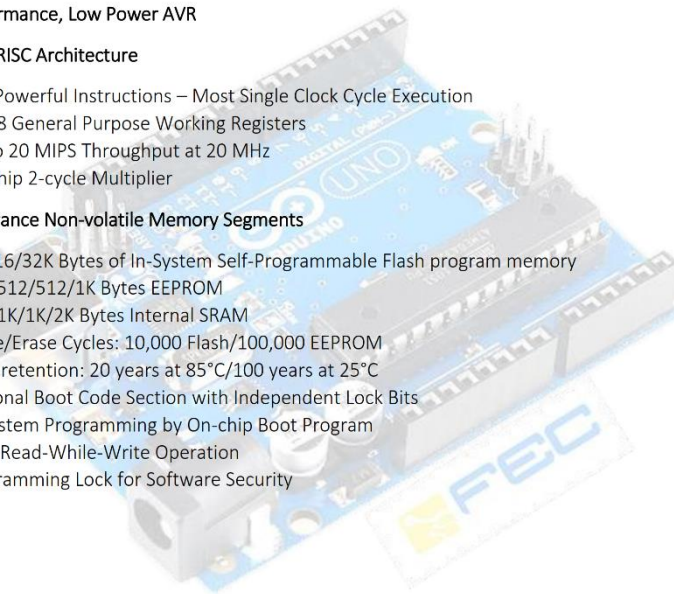


ARDUINO UNO-R3 PHYSICAL COMPONENTS

ATMEGA328P-PU microcontroller

The most important element in Arduino Uno R3 is ATMEGA328P-PU is an 8-bit Microcontroller with flash memory reach to 32k bytes. It's features as follow:

- High Performance, Low Power AVR
- Advanced RISC Architecture
 - 131 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Up to 20 MIPS Throughput at 20 MHz
 - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
 - 256/512/512/1K Bytes EEPROM
 - 512/1K/1K/2K Bytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Six PWM Channels
 - 8-channel 10-bit ADC in TQFP and QFN/MLF package
 - Temperature Measurement
 - 6-channel 10-bit ADC in PDIP Package
 - Temperature Measurement
 - Programmable Serial USART





- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface (Philips I2 C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

• **Special Microcontroller Features**

- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated Oscillator
- External and Internal Interrupt Sources
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby

• **I/O and Packages**

- 23 Programmable I/O Lines
- 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

• **Operating Voltage:**

- 1.8 - 5.5V

• **Temperature Range:**

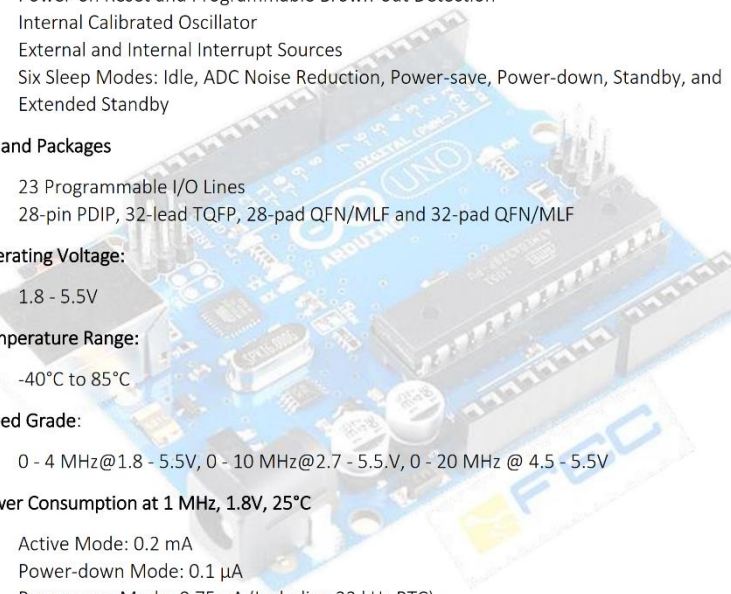
- -40°C to 85°C

• **Speed Grade:**

- 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V

• **Power Consumption at 1 MHz, 1.8V, 25°C**

- Active Mode: 0.2 mA
- Power-down Mode: 0.1 μ A
- Power-save Mode: 0.75 μ A (Including 32 kHz RTC)





- Pin configuration

(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 (\overline{SS} /OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

ATMEGA16u2- mu microcontroller

Is a 8-bit microcontroller used as USB driver in Arduino uno R3 it's features as follow:

- High Performance, Low Power AVR
- Advanced RISC Architecture
 - 125 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
- Non-volatile Program and Data Memories
 - 8K/16K/32K Bytes of In-System Self-Programmable Flash
 - 512/512/1024 EEPROM
 - 512/512/1024 Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/ 100,000 EEPROM
 - Data retention: 20 years at 85°C/ 100 years at 25°C



- Optional Boot Code Section with Independent Lock Bits
- In-System Programming by on-chip Boot Program hardware-activated after reset
- Programming Lock for Software Security
- **USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion**
 - Complies fully with Universal Serial Bus Specification REV 2.0
 - 48 MHz PLL for Full-speed Bus Operation: data transfer rates at 12 Mbit/s
 - Fully independent 176 bytes USB DPRAM for endpoint memory allocation
 - Endpoint 0 for Control Transfers: from 8 up to 64-bytes
 - 4 Programmable Endpoints:
 - IN or Out Directions
 - Bulk, Interrupt and Isochronous Transfers
 - Programmable maximum packet size from 8 to 64 bytes
 - Programmable single or double buffer
 - Suspend/Resume Interrupts
 - Microcontroller reset on USB Bus Reset without detach
 - USB Bus Disconnection on Microcontroller Request
- **Peripheral Features**
 - One 8-bit Timer/Counters with Separate Prescaler and Compare Mode (two 8-bit PWM channels)
 - One 16-bit Timer/Counter with Separate Prescaler, Compare and Capture Mode (three 8-bit PWM channels)
 - USART with SPI master only mode and hardware flow control (RTS/CTS)
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- **On Chip Debug Interface (debug WIRE)**
- **Special Microcontroller Features**
 - Power-On Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Five Sleep Modes: Idle, Power-save, Power-down, Standby, and Extended Standby
- **I/O and Packages**
 - 22 Programmable I/O Lines
 - QFN32 (5x5mm) / TQFP32 packages



OTHER ARDUINO UNO R3 PARTS

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using `pinMode()`, `digitalWrite()`, and `digitalRead()` functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 k Ohms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the `analogWrite()` function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

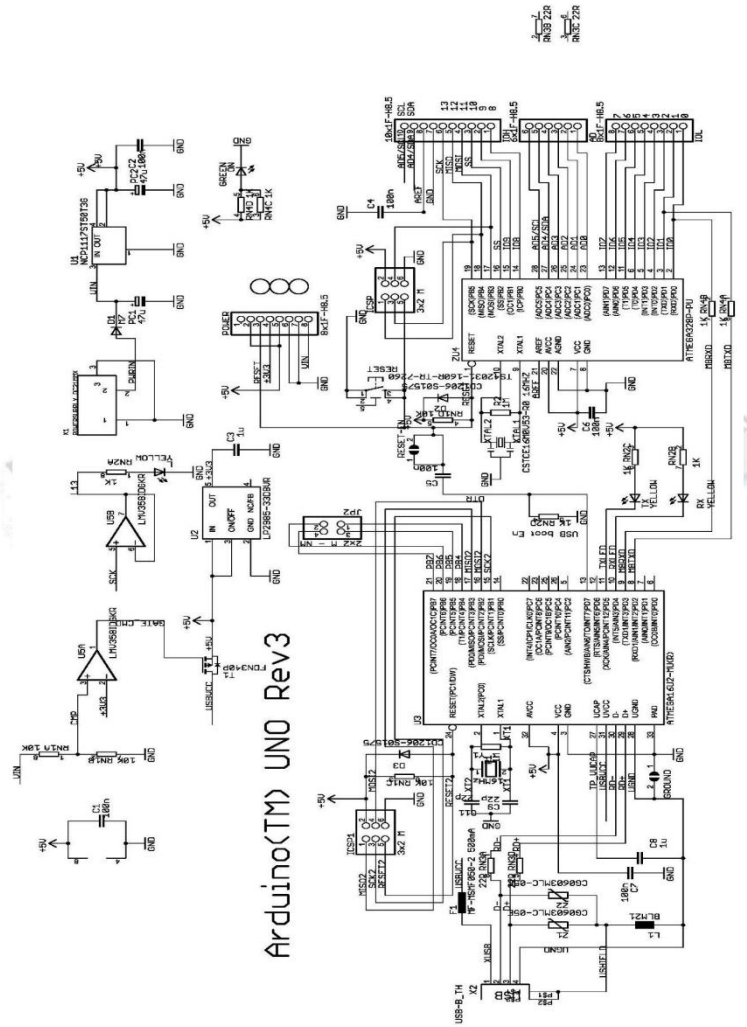
The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the `analogReference()` function. Additionally, some pins have specialized functionality:

- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF: Reference voltage for the analog inputs. Used with `analogReference()`.
- Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

ARDUINO UNO R3 SCHEMATIC DIAGRAM



Lampiran 8. Datasheet Sensor Ultrasonik GY-US42V2 MB 1242


I2CXL-MaxSonar® - EZ™ Series



I2CXL-MaxSonar® - EZ™ Series

High Performance Sonar Rangefinder

MB1202, MB1212, MB1222, MB1232, MB1242⁶

The I2CXL-MaxSonar-EZ series offers industrial ultrasonic sensors with an easy to use I2C interface. These sensors have a high acoustic power output along with real-time auto calibration for changing conditions (voltage and acoustic or electric noise) that ensure users receive the most reliable (in air) ranging data for every reading taken. The low power 3V – 5.5V operation provides very short to long-range detection and ranging, in a compact form factor. The I2CXL-MaxSonar-EZ detect objects from 0-cm to 765-cm (25.1 feet) and provide sonar range information from 20-cm or 25-cm out to 765-cm with 1-cm resolution. Objects from 0-cm* to minimum distance typically range as minimum distance.¹ (*Objects from 0-mm to 1-mm may not be detected). ¹See Close Range Operation*



Features	Low Power Requirement	Applications and Uses
<ul style="list-style-type: none"> I2C bus communication allows rapid control of multiple sensors with only two wires High acoustic power output Real-time auto calibration and noise rejection for every ranging cycle Calibrated beam patterns Continuously variable gain Object presence information as close as 1-mm from the sensor. Range information starting at min. distance. 3V to 5.5V supply with very low average current draw Readings can occur up to every 25mS (40Hz rate)⁴ for up-close objects. 15Hz rate for full range. Triggered operation provides a new range reading as desired Ultrasonic signal frequency of 42KHz Status pin available to determine sensor state Power-up address reset pin available Physical dimensions match other XL-MaxSonar-EZ products Operational temperature from 0°C to +65°C (32F to +149F)⁵ 	<ul style="list-style-type: none"> Wide, low supply voltage requirements eases battery powered design Low current draw reduces current drain for battery operation Fast first reading after power-up eases battery requirements <p>Benefits</p> <ul style="list-style-type: none"> Acoustic and electric noise resistance Reliable and stable range data Low cost Quality controlled beam characteristics Very low power rangefinder, excellent for multiple sensor or battery based systems Ranging is triggered externally Fast measurement cycle No power up calibration required Perfect for when objects may be directly in front of the sensor during power up Easy mounting 	<ul style="list-style-type: none"> Multi-sensor arrays Proximity zone detection People detection Robot ranging sensor Autonomous navigation Educational and hobby robotics Environments with acoustic and electrical noise Distance measuring Long range object detection Security systems Motion detection Landing flying objects Collision avoidance Bin level measurement <p>Notes:</p> <p>¹See Close Range Operation</p> <p>²Users are encouraged to evaluate the sensors performance in their application</p> <p>³By design</p> <p>⁴Recommended time between readings of 100ms (10Hz Rate)</p> <p>⁵Please reference page 6 for minimum operating voltage verses temperature information.</p> <p>⁶Please reference page 19 for part number key.</p>

Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 20cm. Although most users find MaxSonar sensors to work reliably for detecting objects from 0 to 20cm in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

MaxBotix® Inc.
Copyright 2005 - 2012 MaxBotix Incorporated
Patent 7,679,996

MaxBotix Inc., products are engineered and assembled in the USA

Page 1
Web: www.maxbotix.com
PD11648h

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor determines the range to a target.

I2CXL-MaxSonar-EZ Pin Out

GND: Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.

V+: Operates on 3VDC to 5.5VDC. The average current draw for 3.3V operation is 2.7mA (50mA peak) and for 5V operation is 4.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit. Please reference page 6 for minimum operating voltage verses temperature information.

Pin 5-SCL (I2C Clock): This is the clock line for I2C communications. These sensors support I2C clock frequencies up to 400kHz provided clock stretching is supported by the master device. Without clock stretching the devices can run at speeds up to 50kHz.

Pin 4-SDA (I2C Data): This is the data line for I2C communications. These sensors operate as I2C slave devices.

Pin 3-Not Used: This pin is not used.

Pin 2-Address Announce / Status: While the sensor is performing a range reading, this pin is set high and I2C communications are ignored. During non-ranging operation, this pin is held low and the sensor is listening for incoming I2C communication. Optionally, users may poll this pin to determine if the sensor has finished its ranging cycle and is ready to report the latest range information.

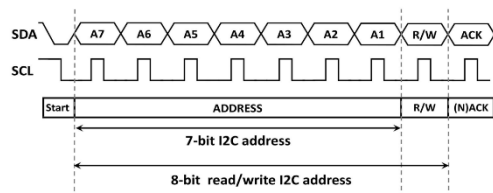
During power-up this pin will provide a pulse width representation of the sensors current address with a length of ~100 microseconds per digit. (The default address of 224 will announce with a pulse of 22,400 microseconds in length)

Pin 1-Temporary Default: This pin is internally pulled high. On power up, the state of this pin is checked; if left high or disconnected, the sensor will use the address stored memory for I2C communications. If pulled low, the sensor will use its default address for the current power cycle.

I2CXL-MaxSonar-EZ Default Address

The representation of the sensor address will be different depending on the addressing scheme your master device uses. The chart below shows the default address for the I2C-MaxSonar-EZ sensors under different addressing implementations. Elsewhere in this datasheet a 8-bit read/write addressing scheme is assumed.

Addressing Scheme	Default Address (decimal)	Default Address (binary)	Notes
7-bit addressing	112	1110 000X	7-bit addressing handles the address shifting and R/W bit for the user
8-bit addressing	224	1110 000X	8-bit addressing inserts the R/W bit and only allows for even number addresses
8-bit read/write addressing	Write: 224 Read: 225	1110 0000 1110 0001	8-bit R/W addressing schemes require the user to set the R/W bit directly.



I2C-MaxSonar-EZ

MaxBotix® Inc.
Copyright 2005 - 2012 MaxBotix Incorporated
Patent 7,679,996

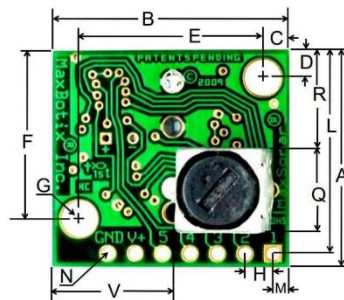
MaxBotix Inc., products are engineered and assembled in the USA

Commands

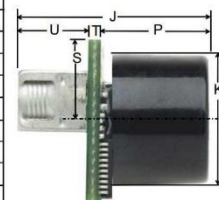
Page 2
Web: www.maxbotix.com
PD11848h

Command	Sequence of Events	Value Used (decimal)	Value Used (binary)	Notes
Take a range reading	1. Initiate a write at the sensor address 2. Write the range command byte	224 (default) 81	1110 0000 0101 0001	Commands the sensor to take a single range reading and save to distance found for the next range request. It is best to allow 100ms between readings to allow for proper acoustic dissipation.
Report the last range value	1. Initiate a read at the sensor address 2a. Read the two bytes from the sensor starting with the range high byte. 2b. Read the range low byte.	225 (default) <i>(Sent by sensor)</i> <i>(Sent by sensor)</i>	1110 0001 Range High Byte Values are MSB to LSB	The sensor will report the distance value in cm obtained from its last range reading. Users requiring real-time information should command a range reading ~80ms before reading the sensor. After power-up if no range command is sent the sensor respond with two part info bytes.
Change the sensor address	1. Initiate a write at the sensor address 2a. Write three bytes to the sensor starting with the addr_unlock_1 command 2b. Write the addr_unlock_2 command 2c. Write the new sensor address	224 (default) 170 165 <i>(User Value)</i>	1110 0000 1010 1010 1010 0101 ##### ##0	The sensor will only accept even address values. If an odd numbered address is sent the sensor will be set to the next lowest even number. If the sensor is told to change to one of the invalid addresses below the sensor will ignore this command and stay at its current address. Invalid Address Values: 0, 80, 164, 170

I2C-MaxSonar-EZ Mechanical Dimensions

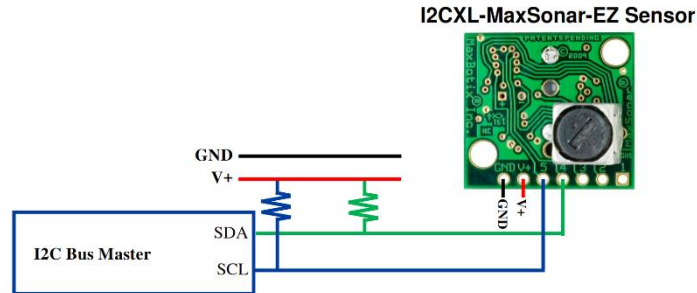


A	0.785"	19.9mm	L	0.735"	18.7mm
B	0.870"	22.1mm	M	0.065"	1.7mm
C	0.100"	2.54mm	N	0.038" <small>as</small>	1.0mm <small>as</small>
D	0.100"	2.54mm	P	0.537"	13.64mm
E	0.670"	17.0mm	Q	0.304"	7.72mm
F	0.610"	15.5mm	R	0.351"	8.92mm
G	0.124" <small>as</small>	3.1mm <small>as</small>	S	0.413"	10.5mm
H	0.100"	2.54mm	T	0.063"	1.6mm
J	0.989"	25.11mm	U	0.368"	9.36mm
K	0.645"	16.4 mm	V	0.492"	12.5mm
values are nominal			Weight, 5.9 grams		



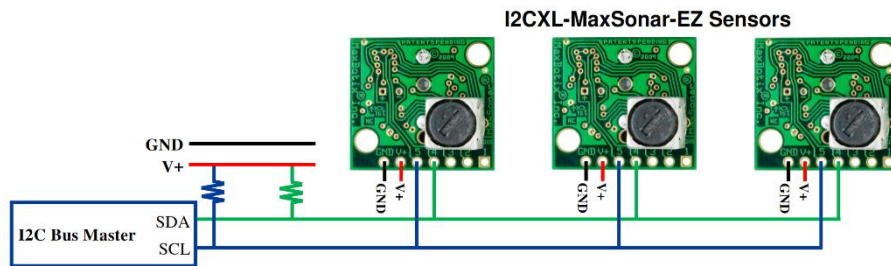
Single Sensor Wiring Diagram

The I2C bus is a two wire interface that consists of a clock line and data line where each requires a pull-up resistor attached to V+. Only one pull-up resistor is required each for the SCL and SDA lines per bus – not per sensor.



The I2C specification recommends a resistance value of 4.7 k Ω for 20-100kHz interfaces with good low inductance routing. However, these specifications are for communication between chips on a single PCB. If you have longer cable lengths it is best to use lower value resistor, such as 1k Ω , and also to use properly shielded cables. Often I2C bus problems can be fixed by doing one of the following: by using properly shielded cable or by decreasing the value of the pull-up resistors. The I2CXL-MaxSonar-WR/WRC series is capable of sinking more current than the I2C specification requires (15mA versus 3mA) so a much lower resistance value can be used. The voltage applied to the I2C lines should be the same voltage that is applied to V+ of the sensor.

Multiple Sensor Wiring Diagram



Selecting a I2CXL-MaxSonar-EZ Sensor

Different applications require different sensors. The I2CXL-MaxSonar-EZ product line offers varied sensitivity to allow users to select the best sensor to meet their needs.

People Detection Wide Beam High Sensitivity	Best Balance	Large Targets Narrow Beam Noise Tolerance	Very Large Targets Narrow Beam Extreme Noise Tolerance
MB1202	MB1212	MB1222	MB1232
			MB1242

The diagram above shows how each product balances sensitivity and noise tolerance. This does not affect the pin outputs or other sensor operation. To get a better view of how each sensor will function to targets of different sizes reference the I2CXL-MaxSonar-EZ beam patterns.

Sensor Minimum Distance

For the MB1202 and MB1212 the sensor minimum reported distance is 25-cm. However, the I2CXL-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 25-cm will typically range as 25-cm.

For the MB1222, MB1232, & MB1242 the sensor minimum reported distance is 20-cm. However, the I2CXL-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 20-cm will typically range as 20-cm.

Real-time Noise Rejection

While the I2CXL-MaxSonar®-EZ is designed to operate in the presence of noise, best operation is obtained when noise strength is low and desired signal strength is high. The user is encouraged to mount the sensor in such a way that minimizes outside acoustic noise pickup. In addition, keep the DC power to the sensor free of noise. This will let the sensor deal with noise issues outside of the users direct control (in general, the sensor will still function well even if these things are ignored). Users are encouraged to test the sensor in their application to verify usability.

For every ranging cycle, individual filtering for that specific cycle is applied. In general, noise from regularly occurring periodic noise sources such as motors, fans, vibration, etc., will not falsely be detected as an object. This holds true even if the periodic noise increases or decreases (such as might occur in engine throttling or an increase/decrease of wind movement over the sensor). Even so, it is possible for sharp non-periodic noise sources to cause false target detection. In addition, *(because of dynamic range and signal to noise physics,) as the noise level increases, at first only small targets might be missed, but if noise increases to very high levels, it is likely that even large targets will be missed.

The I2CXL-MaxSonar-EZ series has additional resistance against periodic noise and small target rejection capabilities over the standard XL-MaxSonar-EZ/AE series.

*In high noise environments, if needed, use 5V power to keep acoustic signal power high. In addition, a high acoustic noise environment may use some of the dynamic range of the sensor, so consider a part with less gain such as the MB1222, MB1232, or MB1242. For applications with large targets, consider a part with clutter rejection like the MB7360.

Target Size Compensation

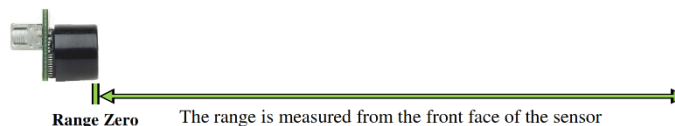
The I2CXL-MaxSonar-EZ sensors do not apply target size compensation to detected objects. This means that larger size targets may report as a closer distance than the actual distance or smaller size targets may report as a further distance than the actual distance.

Real-Time Auto Calibration

Each time before the I2CXL-MaxSonar-EZ sensor takes a range reading it calibrates itself. The sensor then uses this data to range objects. If the humidity or applied voltage changes during sensor operation, the sensor will continue to function normally. **The sensor does not apply compensation for the speed of sound change due to temperature to any range readings.** (20C operation is assumed when calculating the speed of sound, other MaxBotix Inc., products are available with these features)

Range “0” Location

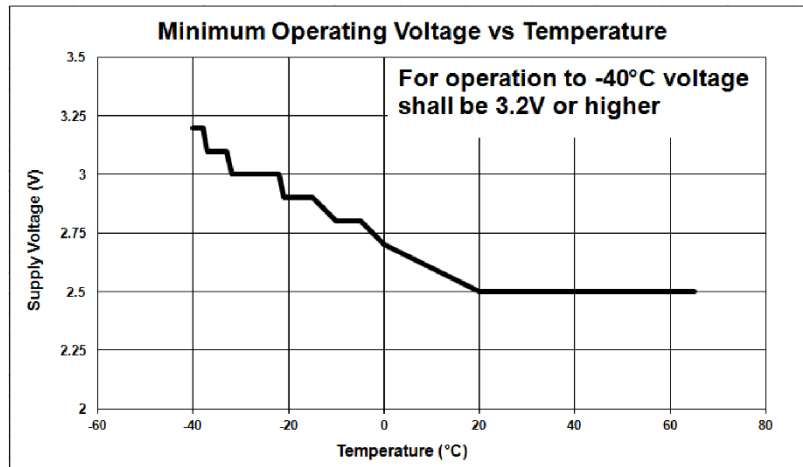
The I2CXL-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.



The I2CXL-MaxSonar-EZ will report the range to the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Voltage vs Temperature

The graph below shows minimum operating voltage of the sensor verses temperature.



MB1242: I2CXL-MaxSonar-EZ4

The I2CXL-MaxSonar-EZ4 has the highest noise tolerance and the narrowest beam of any of our indoor sensors. The sensor is calibrated and tested to provide stable range readings to large targets even in electrically and acoustically noisy environments.

MB1242-000 MB1242-040 I2CXL-MaxSonar®-EZ4™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor.

A 6.1-mm (0.25-inch) diameter dowel

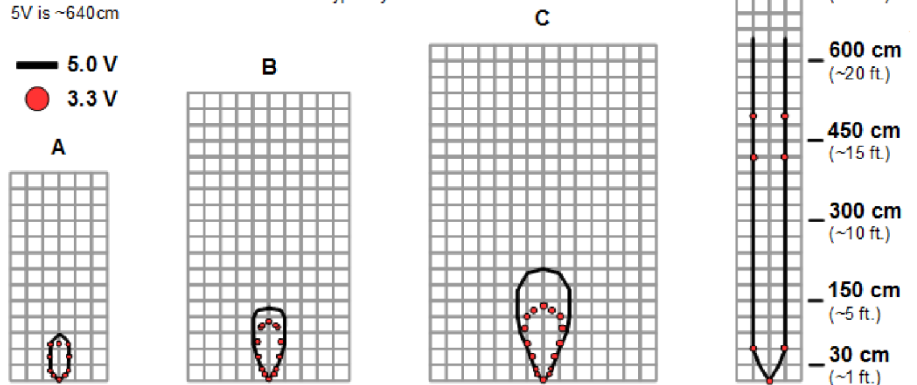
B 2.54-cm (1-inch) diameter dowel

C 8.89-cm (3.5-inch) diameter dowel

Note: The maximum detected distance for 3.3V is ~500cm and for 5V is ~640cm

D 11-inch wide board moved left to right with the board parallel to the front sensor face. This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Patterns drawn to a 1:95 scale for easy comparison to our other products.

MB1242 Features and Benefits

- Narrowest beam sensor in I2CXL-MaxSonar-EZ line
- Low power consumption
- Easy to use interface
- Rejects small clutter
- Can be powered by many different types of power sources

MB1242 Applications and Uses

- Security
- Motion detection
- Landing flying objects
- Useable with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance

I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```

/*          Arduino I2C for a MaxSonar          */
// Arduino I2C for a MaxSonar by Carl Myhre is licensed under a
// Creative Commons Attribution-ShareAlike 4.0 International License.
// Original Author: Carl Myhre, 10-02-2014, Revision: 1.0
// Modifications by:
//
// Revision History: 1.0 -- 10-02-2014 -- Created initial code build
//
// The original I2C libraries were created by Peter Fleury
// http://homepage.hispeed.ch/peterfleury/avr-software.html
//
// These libraries were adapted by Bernhard Nebel for use on Arduino
// https://github.com/felias-fogg/SoftI2CMaster
//
// Special Thanks to MaxBotix Inc. for sponsoring this project!
// http://www.maxbotix.com -- High Performance Ultrasonic Sensors
//
// For more information on installing the I2C libraries for Arduino
// visit http://playground.arduino.cc/Main/SoftwareI2CLibrary
//
//Hints on installing this code:
// 1. You will need to install the <SoftI2CMaster.h> library before using this code.
//    On Windows, the files are placed in C:\Program Files (x86)\Arduino\libraries\SoftI2CMaster\
// 2. As of 10-02-14 the Arduino library page (reference above) has the wrong name for the include file
//    it lists <SoftI2C.h> instead of <SoftI2CMaster.h> -- use the one that matches your installation.
// 3. Make sure to load the library into the Arduino compiler.
//    To do this go to: SKETCH >> IMPORT LIBRARY... >> ADD LIBRARY...
//    Then navigate to C:\Program Files (x86)\Arduino\libraries\SoftI2CMaster\SoftI2CMaster.h
// 4. Be sure to set the SCL and SDA pins so that they match the pins you are using.
// 5. I have included 3 working "code examples" which differ from the 3 "functions" I included.
//    The functions are all that should be required to quickly use the I2C library to talk to a MaxSonar.
//    The three code examples show how I would implement each of the common tasks you may wish to do.
// 6. The included functions are as follows:
//    A. start_sensor(addr)
//    B. read_sensor(addr)
//    C. change_address(oldaddr,newaddr)
// 7. The included code examples are as follows:
//    A. read_the_sensor_example()
//    B. address_polling_example()
//    C. default_address_change_example()
// 8. You do not have to keep track of the error codes passed out by the installed functions if you do not want to.
//    I included the error tracking so that it was easy for others to build a reliable system -- and to ease
//    troubleshooting. (not using it makes for cleaner code if you trust your interface)

```


I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```

/*
Below, I define the SCL and SDA pins by their ATMEGA pins I have included links to common mappings below.
  UNO:  http://arduino.cc/en/Hacking/PinMapping168
  NANO: (matches UNO but has fewer pins)
  MEGA 2560: http://arduino.cc/en/Hacking/PinMapping2560
The current data matches the setup for the Arduino Uno -- they may need to be changed if the hardware changes.
You can also switch the I2C interface
to any of the tristate pins that you want (not just the SDA or SCL pins).
*/
#define SCL_PIN 5           //Default SDA is Pin5 PORTC for the UNO -- you can set this to any tristate pin
#define SCL_PORT PORTC
#define SDA_PIN 4           //Default SCL is Pin4 PORTC for the UNO -- you can set this to any tristate pin
#define SDA_PORT PORTC
#define I2C_TIMEOUT 100    //Define a timeout of 100 ms -- do not wait for clock stretching longer than this time

/*
I have included a couple of extra useful settings for easy reference.
//define I2C_CPUFREQ (F_CPU/8) //Useful if you plan on doing any clock switching
#define I2C_FASTMODE 1       //Run in fast mode (400 kHz)
#define I2C_SLOWMODE 1      //If you do not define the mode it will run at 100kHz with this define set to 1 it
                             //will run at 25kHz
*/
#include <SoftI2CMaster.h>    //You will need to install this library

void setup(){
  // Initialize both the serial and I2C bus
  Serial.begin(9600);
  i2c_init();

  // (OPTIONAL) Check each address for a sensor
  address_polling_example();

  /*
  Note that I placed the address change example in setup() for a good reason.
  Changing the sensor address causes an EEPROM write, there should only be ~1,000,000+
  of these writes to the sensor microcontroller over its product lifetime.
  Changing the address is fine, but doing it every second for the next 4 years may
  cause reliability issues.
  */
  // (OPTIONAL) Run an address change example
  default_address_change_example();

  // Your code here
}

void loop()
{
  // (OPTIONAL) Read a sensor at the default address
  read_the_sensor_example();

  // Your code here
}

```

I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```

////////////////////////////////////
// Function: Start a range reading on the sensor //
////////////////////////////////////
//Uses the I2C library to start a sensor at the given address
//Collects and reports an error bit where: 1 = there was an error or 0 = there was no error.
//INPUTS: byte bit8address = the address of the sensor that we want to command a range reading
//OUTPUTS: bit errorlevel = reports if the function was successful in taking a range reading: 1 = the function
//          had an error, 0 = the function was successful
boolean start_sensor(byte bit8address){
    boolean errorlevel = 0;
    bit8address = bit8address & B11111110;          //Do a bitwise 'and' operation to force the last bit to be
                                                    //zero -- we are writing to the address.
    errorlevel = !i2c_start(bit8address) | errorlevel; //Run i2c_start(address) while doing so, collect any errors
                                                    //where 1 = there was an error.
    errorlevel = !i2c_write(81) | errorlevel;         //Send the 'take range reading' command. (notice how the
                                                    //library has error = 0 so I had to use "!" (not) to invert
                                                    //the error

    i2c_stop();
    return errorlevel;
}

////////////////////////////////////
// Function: Read the range from the sensor at the specified address //
////////////////////////////////////
//Uses the I2C library to read a sensor at the given address
//Collects errors and reports an invalid range of "0" if there was a problem.
//INPUTS: byte bit8address = the address of the sensor to read from
//OUTPUTS: int range = the distance in cm that the sensor reported; if "0" there was a communication error
int read_sensor(byte bit8address){
    boolean errorlevel = 0;
    int range = 0;
    byte range_highbyte = 0;
    byte range_lowbyte = 0;
    bit8address = bit8address | B00000001; //Do a bitwise 'or' operation to force the last bit to be 'one' -- we are
                                                    //reading from the address.
    errorlevel = !i2c_start(bit8address) | errorlevel;
    range_highbyte = i2c_read(0); //Read a byte and send an ACK (acknowledge)
    range_lowbyte = i2c_read(1); //Read a byte and send a NACK to terminate the transmission
    i2c_stop();
    range = (range_highbyte * 256) + range_lowbyte; //compile the range integer from the two bytes received.
    if(errorlevel){
        return 0;
    }
    else{
        return range;
    }
}

```

I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```

////////////////////
// Function: Change the sensor address //
////////////////////
//Uses the I2C library to change the address of a sensor at a given address
//Collects and reports an error bit where: 1 = there was an error or 0 = there was no error.
//INPUTS: byte oldaddress = the current address of the sensor that we want to change
//INPUTS: byte newaddress = the address that we want to change the sensor to
//OUTPUTS: bit errorlevel = reports if the function was successful in changing the address: 1 = the function had an
//          error, 0 = the function was successful
boolean change_address(byte oldaddress,byte newaddress){
    //note that the new address will only work as an even number (odd numbers will round down)
    boolean errorlevel = 0;
    oldaddress = oldaddress & B11111110; //Do a bitwise 'and' operation to force the last bit to be zero -- we are
                                         //writing to the address.
    errorlevel = !i2c_start(oldaddress) | errorlevel; //Start communication at the new address and track error codes
    errorlevel = !i2c_write(170) | errorlevel;         //Send the unlock code and track the error codes
    errorlevel = !i2c_write(165) | errorlevel;         //Send the unlock code and track the error codes
    errorlevel = !i2c_write(newaddress) | errorlevel; //Send the new address
    i2c_stop();
    return errorlevel;
}

////////////////////
// Code Example: Read the sensor at the default address //
////////////////////
void read_the_sensor_example(){
    boolean error = 0; //Create a bit to check for catch errors as needed.
    int range;

    //Take a range reading at the default address of 224
    error = start_sensor(224); //Start the sensor and collect any error codes.
    if (!error){               //If you had an error starting the sensor there is little point in reading it as you
                               //will get old data.

        delay(100);
        range = read_sensor(224); //reading the sensor will return an integer value -- if this value is 0 there was
                                   //an error
        Serial.print("R:");Serial.println(range);
    }
}

```

I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```

////////////////////////////////////
// Code Example: Poll all possible addresses to find a sensor //
////////////////////////////////////
void address_polling_example(){
  boolean error = 0; //Create a bit to check for catch errors as needed.
  int range = 0;
  Serial.println("Polling addresses...");

  //Walk through all possible addresses and check for a device that can receive the range command and will
  //  return two bytes.
  for (byte i=2; i!=0; i+=2){ //start at 2 and count up by 2 until wrapping to 0. Checks all addresses (2-254)
                              //except 0 (which cannot be used by a device)

    error = 0;
    error = start_sensor(i); //Start the sensor and collect any error codes.
    if (!error){             //If you had an error starting the sensor there is little point in reading it.
      delay(100);
      range = read_sensor(i); //reading the sensor will return an integer value -- if this value is 0 there was
                              //an error

      Serial.println(i);
      if (range != 0){
        Serial.print("Device found at:");Serial.print(i);Serial.print(" Reported value of:");Serial.println(range);
      }
    }
    else{
      Serial.print("Couldn't start:");Serial.println(i);
    }
  }

  Serial.println("Address polling complete.");
}

////////////////////////////////////
// Code Example: Change the default address //
////////////////////////////////////
void default_address_change_example(){
  boolean error = 0; //Create a bit to check for catch errors as needed.
  int range;

  Serial.println("Take a reading at the default address");

  //Take a range reading at the default address of 224
  error = start_sensor(224); //Start the sensor and collect any error codes.
  if (!error){               //If you had an error starting the sensor there is little point in reading it.
    delay(100);
    range = read_sensor(224); //reading the sensor will return an integer value -- if this value is 0 there was
                              //an error
    Serial.print("R:");Serial.println(range);
  }

  Serial.println("Change the sensor at the default address to 222");
}

```

I2C Code Examples

Arduino Uno (as of Arduino 1.0.6)

```
//Change the address from 224 to 222
error = 0;
error = change_address(224,222); //Change the address -- I don't do anything with the error handler at this point
//but you can if you want.
delay(200); //Wait 125ms for the sensor to save the new address and reset

Serial.println("Take a reading at the new address");

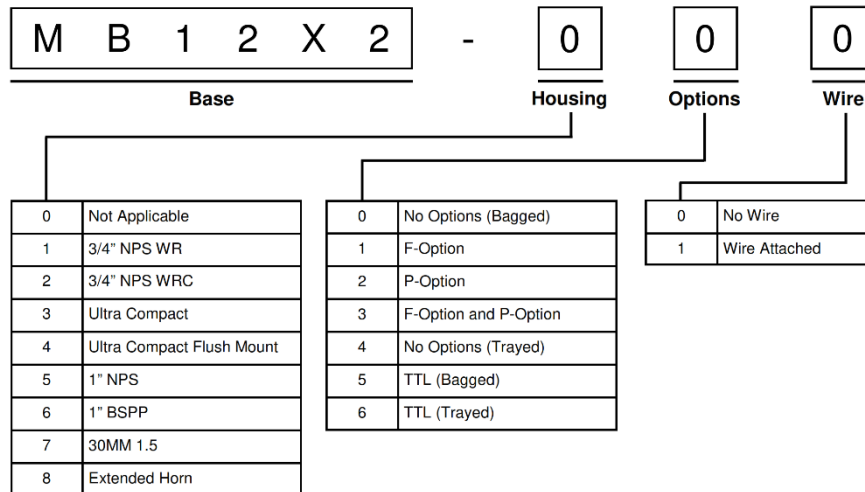
//Take a range reading at the new address of 222
error = 0;
error = start_sensor(222); //Same as above but at the new address
if (!error){
    delay(100);
    range = read_sensor(222);
    Serial.print("N:");Serial.println(range);
}

Serial.println("Change the sensor back to the default address");

//Change the address from 222 to 224
error = 0;
error = change_address(222,224);
delay(200); //Wait 125ms for the sensor to save the new address and reset
}
```

Part Numbers

All part numbers are a combination of a six-character base followed by a dash and a three-digit product code. Please review the following table for more information on the three-digit product code.



The following table displays all of the active and valid part numbers for this product.

Active Part Numbers for MB1202, MB1212, MB1222, MB1232 and MB1242				
MB1202-000	MB1212-000	MB1222-000	MB1232-000	MB1242-000
MB1202-040	MB1212-040	MB1222-040	MB1232-040	MB1242-040

Lampiran 9. Datasheet Motor Servo SG90

