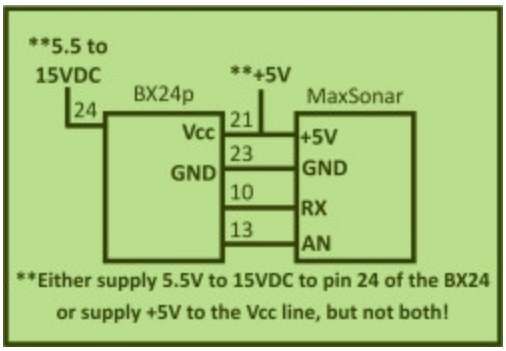
Code Example for BasicX, BX24p




```
BX24
'MaxSonar®-EZ1™ Code Example
'By Chris Harriman
'01/09/2006
'The program below continues to read the MaxSonar®-EZ1™
'It uses the AD to read, and debug to output the data.Const RX As Byte = 10
Const AN As Byte = 13
Dim AnalogOutPut As Byte
Dim SerialOutPut As Byte
Dim PWOutPut As Byte "*******************************************
Sub Main()
Do
AnalogOutPut = RangeA      ' Get the Range
Debug.Print "Analog " & CStr(AnalogOutPut)  ' Print the Range
Call SLeep(512)
Loop
End Sub"*******************************************
Function RangeA() As Byte
' Reads the Analog output of the MAXSonar® EZ1™ (AN Pin) and returns the target
range as a Byte
Dim AValue As Integer
Call PutPin (RX, 0)    ' Turn off the EZ1™ just in case we started with it on
Call PutPin (RX, 1)  ' Turn on the EZ1™
Call Sleep(40)   ' Wait about 50 ms

AValue = GetADC(AN)  ' Read the ADC
RangeA = Cbyte(AValue \ 2)  ' Convert value to Byte and return
End Function
```

Code example for the Basic Micro, Atom

```
‘BasicAtom Code
‘Reads MaxSonar®-EZ1™
‘Bob Gross
’01/14/2005
‘5V connect to +5V
‘GND connect to common
‘TX, connect to Atom P7
‘RX, connect to Atom P5
‘The serial data will be sent when the reading is complete,
‘This is very fast when an object is close.
‘Only four lines of code required

RS232Data var byte                              ‘Set up variable to hold the data
High P5                                         ‘Hold high to start the reading
SerIn P7, n9600, [WAIT(“R”), dec RS232Range ]   ‘wait for “R” and get the data
Low P5                                          ‘Set low when complete
```

Code example using DevBoard-M32 (AVR using Bascom)

```
'Using the MaxSonar®-EZ1™ with the DevBoard-M32
'By Eddy Wright, Wright Hobbies Robotics, 2006
'http://www.wrighthobbies.net

'We will read both the analog and serial outputs of the MaxSonar®

'The M32 A/D converter has an internal voltage
'Reference of 2.56v which is perfect for output
'of the MaxSonar® – 2.55v

'The analog output (AN)of the MaxSonar® is connected to
'Port A.0, the ADC Channel 0
'The serial output (TX) is connected to to Port D.7
'Each loop, we read the analog and serial values

'This code can be used with any AVR with ADC that is supported by Bascom

Dim Dist As Word , Strdist As String * 8 , Serdist As Byte

'Config the softare UART, we need to use the INVERTED option with the MaxSonar®
Open "comd.7:9600,8,n,1,INVERTED" For Input As #1

'Configure ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal

Start Adc

Do
Dist = Getadc(0)
Shift Dist , Right , 2       'The M32 has 10bit ADC, shifting it twice makes it 8bit
Input #1 , Strdist Strdist = Right(strdist , 3)    'Strip off the letter R
Serdist = Val(strdist)     'Convert to a number
Print "Analog Distance = " ; Dist ; " Inches"
Print "Serial Distance = " ; Serdist ; " Inches"
Loop
```

Code Example using Parallax Basic Stamp BS2, BasicStamp

```
‘Reads both the PW and serial outputs
‘ {$STAMP BS2}
‘ {$PBASIC 2.5}
‘ www.danderrick.com/
‘ permission for unlimited use granted to all

‘ First test of the MaxSonar®-EZ1™
‘ micro µ

‘ —- P PINs ————————
pMaxRecv PIN 15
pMaxClock PIN 14
pMaxPWM PIN 0

‘ ——— X Variables ———
xDist VAR Word
xPulse VAR Word
xX VAR Byte

‘ ============ Main loop ================
DO GOSUB sPWM
GOSUB sSerial
DEBUG CR, CR
PAUSE 50
LOOP
END  ‘ never reached

‘

———- Subs ——————

sPWM: ‘Max sends 147 µs per inch
‘BS2 reads for 2 µs
FOR xX = 1 TO 5
HIGH pMaxClock
PULSIN pMaxPWM, 1, xPulse
LOW pMaxClock
DEBUG DEC5 xPulse, ” “
PAUSE 50
NEXT
DEBUG CR
RETURN

sSerial:
FOR xX = 1 TO 5
SERIN pMaxRecv\pMaxClock, 16468, [WAIT (“R”), DEC xDist] DEBUG DEC5 xDist, ” “
PAUSE 50
NEXT
DEBUG CR
RETURN
‘ —— Physical end of file ———
```

C Program Driver for MaxBotix MB7066 PW Ultrasonic Sensor

By Paul Sfetku, July 2011
Development System – Mikroelektronika
http://www.mikroe.com/
Develop. Board: BigPic6 with GLCD (128×64)
Microcontroller: PIC18F8520/8722 family
Compiler: mikroC PRO for PIC
Interface: BigPic6 SW7 RG0: PORTG.B0 (output) → MB7066<4> start/stop ranging
BigPic6 SW7 RG1: PORTG.B1 (input) → MB7066<2> PW – pulse width

NOTE: BigPic6 SW7 is set for pull-up on PORTG pins RG0/B0 & RG1/B1.

MB7066 Pin Out
Pin 2 – PW: This pin outputs a pulse width representation of range. To calculate distance, you can a scale factor of 58us/cm.
Pin 4 – RX: This pin is internally pulled high. The MB7066 will continually measure range and output if the pin is left unconnected or held high. If held low the MB7066 will stop ranging. Bring pin high for >= 20uS for range reading.
Pin 6 – Vcc (3V – 5.5V): Average and peak current draw for 3.3V operation is 2.1mA (50mA peak). 5V operation is 3.4mA (100mA peak) respectively. Peak current is used during sonar pulse transmit.
Pin 7 – GND: Return for the DC power supply. GND (& V+) must be ripple and noise free for best operation.
NOTE: Refer to the MB7066 PW documentation for an explanation of MB7066 Real-time Operation & Timing

Program Code Explanation
The C code listed below was generated using the mikroC Pro PIC compiler. The code will drive the MaxBotix MB7066 pulse width (PW) sensor, and any MaxBotix pulse width sensor.

The MB7066 is operated in free running mode with MB7066<4> pulled High. To operate the MB7066 in start/stop mode, you can uncomment code statements 8,9,10 and code statements 17,18,19.

Statements 15-20 can be used to determine when and if the MB7066 fails to detect a return pulse from the given target. A value for maxvalue in statement 15 is needed to calibrate the sensor for a maximum time or maximum distance value.

The code to drive the MB7066 is contained within an infinite loop declared in statement 11. The MB7066 operates on a 100ms PW cycle. MB7066 pin-2 PW sets high to start a ranging cycle, and sets Low if and when a target object is detected. If no target is detected, pin-2 PW will be held high for up to 62ms or 1068cm, the maximum range value.

Code statements 12-25 are designed detect and measure the period of each PW begin/end cycle. Statement 12 detects when the PW goes High. The delay in statement 13 is used to calibrate the sensor's ranging resolution to detect changes at one inch increments. The variable in statement 14 counts the number of delays until the PW signal does low.

Statements 21-24 calculate the range and detect the beginning of the next 100ms PW cycle. Statement 22 calculates the range based on the time counted in statement 14, and based on a constant factor (.5) that can be adjusted to provide an accurate range measurement.

C Program Driver for MaxBotix MB7066 PW Ultrasonic Sensor Cont.

The loop in statement 24 is used to wait for the PW to set High, indicating the beginning of the next 100ms PW ranging cycle.

```
// Declarations ——————————————-
TRISG = 0x02;                // PORTG DDR: RG0 (output), RG1 (input)
unsigned int msec    = 0;    // time counter – microseconds
unsigned int inches   = 0;   // calculated distance
unsigned maxvalue   = 0;   // sensor maximum – echo not detected

void main() {
//PORTG.B0 = 0;            // MB7066<4>: stop ranging – set Low
//PORTG.B0 = 1;           // MB7066<4>: start ranging – set High
//Delay_us(20);            // 20us delay for start of ranging
while (1) {                 // infinite loop
if (PORTG.B1) {              // MB7066<2>: PW – check if High
Delay_us(30);               // delay factor 30us – gives 1 inch resolution
msec++;                     // count time PW is High each microseconds
//if (msec > maxvalue) {   // PW max time 62ms -> echo not detected
// msec = 0;                // clear counter
// PORTG.B0 = 0;            // MB7066<4>: stop ranging – set Low
// PORTG.B0 = 1;           // MB7066<4>: start ranging – set High
// Delay_us(20);            // 20us delay for start of ranging
//}
} else {                    // MB7066<2>: PW -> Low
inches = .5*msec;           // range (inches): target correction factor = .5
msec = 0;                   // clear counter
while (!PORTG.B1);          // wait for MB7066<2> PW to go High -> start 100ms range cycle
}
}
}
```

Arduino I2C Code Example for I2CXL-MaxSonar Products

Development System – Arduino Uno (as of Arduino 1.0.1)
Develop. Board: Arduino Uno R3

```
/* Code for Arduino Uno R3
Assumes the sensor is using the default address
Sensor Connections:
Pin 7 to GND
Pin 6 to 5V
Pin 5 to SCL
Pin 4 to SDA
Requires pull-ups for SCL and SDA connected to 5V to work reliably
*/
#include “Wire.h”
//The Arduino Wire library uses the 7-bit version of the address, so the code example uses 0x70 instead of the 8-bit 0xE0
#define SensorAddress byte(0x70)
//The Sensor ranging command has a value of 0x51
#define RangeCommand byte(0x51)
//These are the two commands that need to be sent in sequence to change the sensor address
#define ChangeAddressCommand1 byte(0xAA)
#define ChangeAddressCommand2 byte(0xA5)

void setup() {
Serial.begin(9600);//Open serial connection at 9600 baud
Wire.begin();//Initiate Wire library for I2C communications with I2CXL-MaxSonar-EZ
}

void loop() {
takeRangeReading();                    //Tell the sensor to perform a ranging cycle
delay(100);                            //Wait for the sensor to finish
word range = requestRange();           //Get the range from the sensor
Serial.print(“Range:”);Serial.println(range);     //Print to the user
}

//Commands the sensor to take a range reading
void takeRangeReading(){
Wire.beginTransmission(SensorAddress);      //Start addressing
Wire.write(RangeCommand);                   //send range command
Wire.endTransmission();                     //Stop and do something else now
}
```

Arduino I2C Code Example for I2CXL-MaxSonar Products Cont.

```
//Returns the last range that the sensor determined in its last ranging cycle in centimeters. Returns 0 if there is no
communication.
word requestRange(){
Wire.requestFrom(SensorAddress, byte(2));
if(Wire.available() >= 2){                    //Sensor responded with the two bytes
byte HighByte = Wire.read();                  //Read the high byte back
byte LowByte = Wire.read();                   //Read the low byte back
word range = word(HighByte, LowByte);         //Make a 16-bit word out of the two bytes for the range
return range;
}
else {
return word(0);                               //Else nothing was received, return 0
}
}

/* Commands a sensor at oldAddress to change its address to newAddress
oldAddress must be the 7-bit form of the address that is used by Wire
7BitHuh determines whether newAddress is given as the new 7 bit version or the 8 bit version of the address
\ If true, if is the 7 bit version, if false, it is the 8 bit version
*/
void changeAddress(byte oldAddress, byte newAddress, boolean SevenBitHuh){
Wire.beginTransmission(oldAddress);           //Begin addressing
Wire.write(ChangeAddressCommand1);            //Send first change address command
Wire.write(ChangeAddressCommand2);            //Send second change address command

byte temp;
if(SevenBitHuh){ temp = newAddress << 1; }    //The new address must be written to the sensor
else    { temp = newAddress;        }         //in the 8bit form, so this handles automatic shifting
Wire.write(temp);                             //Send the new address to change to
Wire.endTransmission();
}
```

As a minor note, in old Arduino IDE's
Wire.read() should be substituted by Wire.receive()
and
Wire.write() should be substituted by Wire.send()