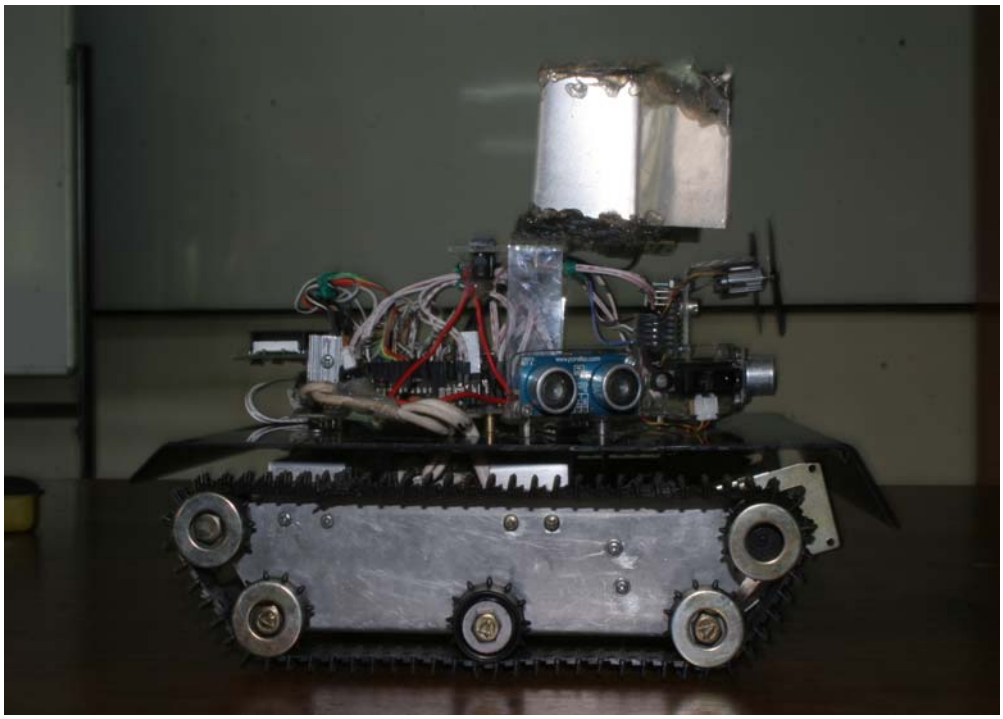
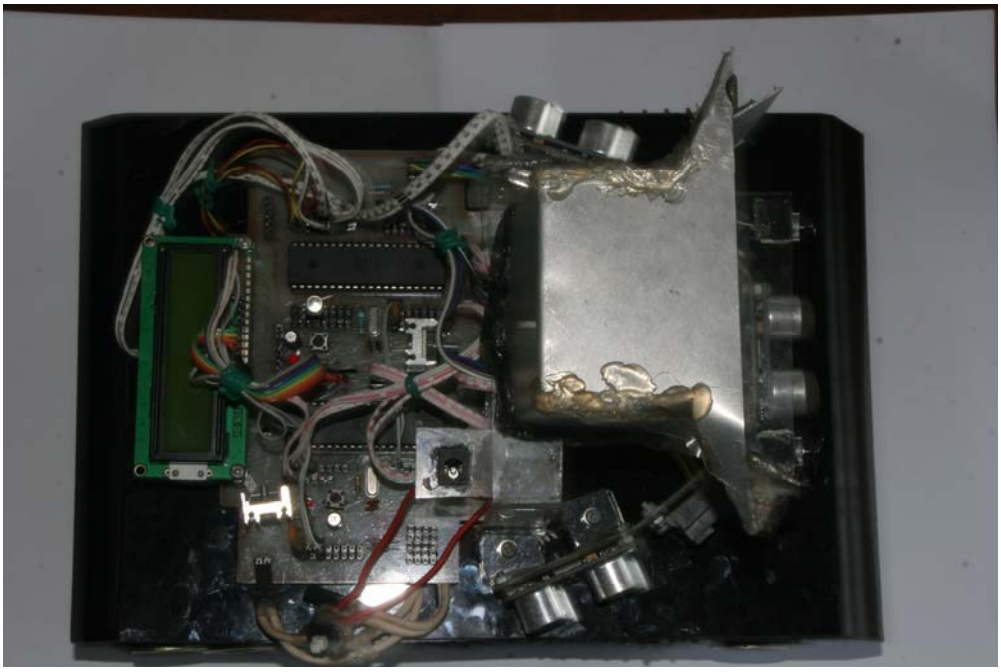
LAMPIRAN A
FOTO ROBOT MOBIL TANK

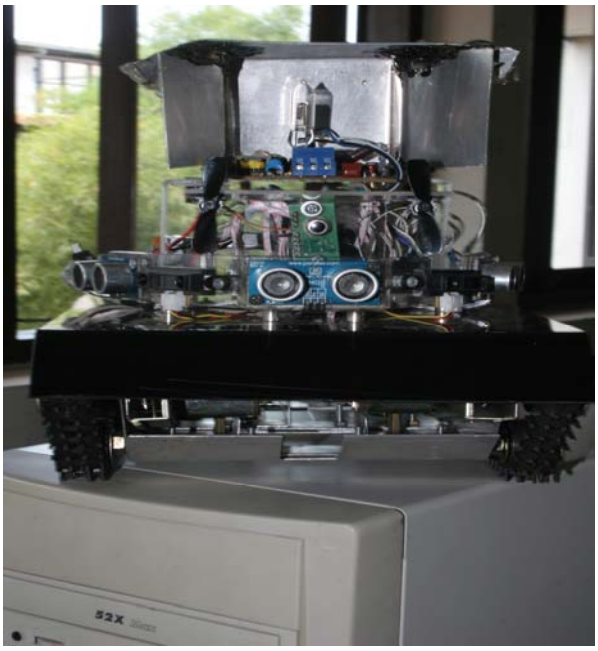
Tampak Samping



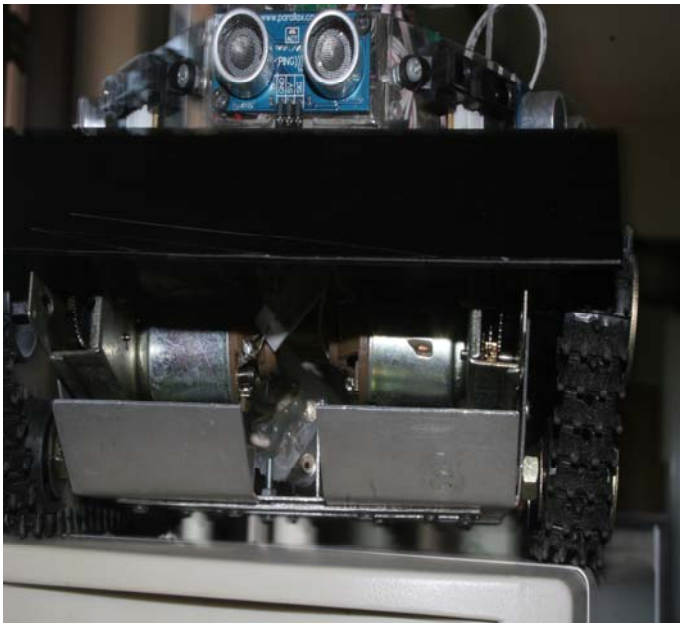
Tampak Atas



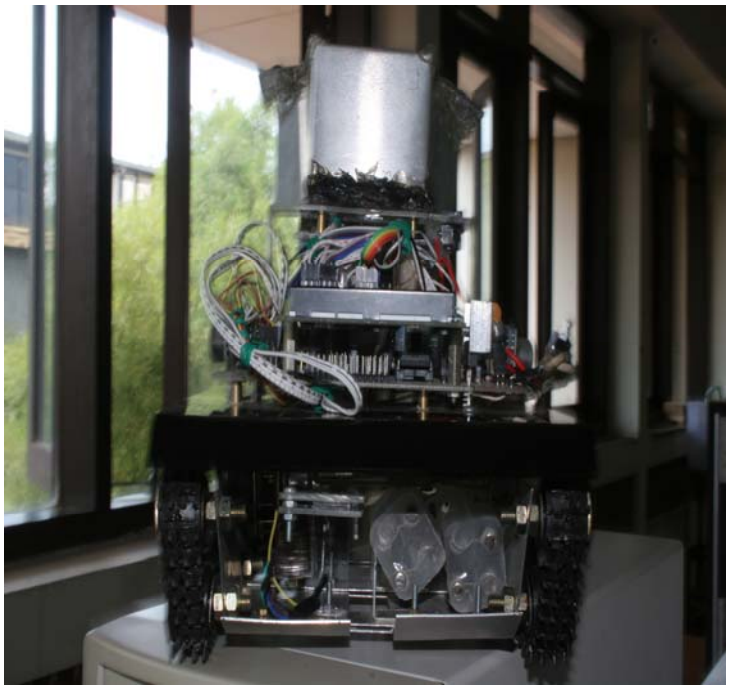
Tampak Depan



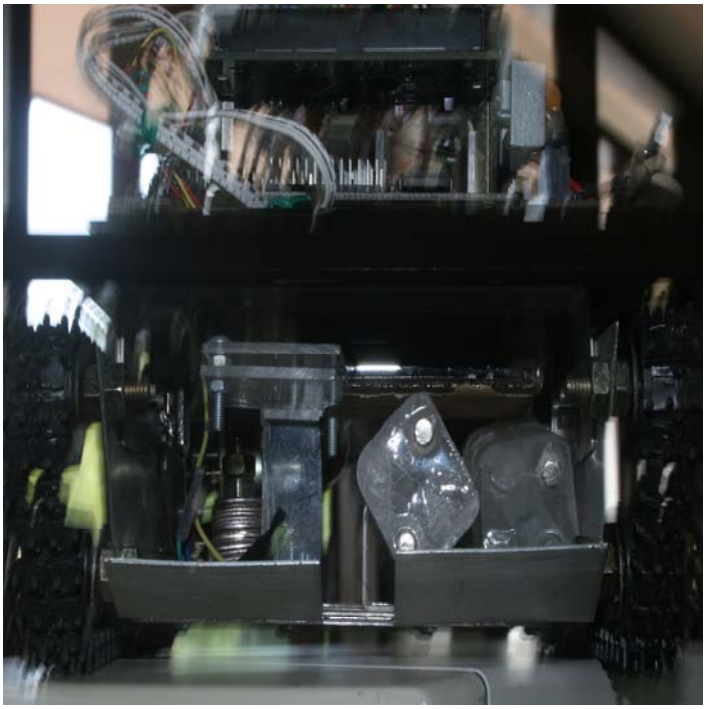
Tampak Depan Bawah



Tampak Belakang



Tampak Belakang Bawah



LAMPIRAN B

PROGRAM PADA PENGONTROL ATMEGA16

PROGRAM UTAMA

MIKROKONTROLER SENSOR

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Professional
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 12/30/2008
Author  : Lab Instrumentasi
Company : UKM
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <pingxx2.h>
#include <delay.h>
#include <math.h>
#include <stdio.h>

// I2C Bus functions
#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm
#include <lcd.h>

// Standard Input/Output functions
#include <stdio.h>

#define ADC_VREF_TYPE 0x60

// Read the 8 most significant bits
// of the AD conversion result
unsigned char read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCH;
}

// Declare your global variables here
```

```
//////// ir
#include <ir_sip2.h>
//////////

/////uvtron
#define flame PINC.7

//////////

/// i2c
eeprom char z=0;

///////TPA81

char data[9],text[32];
//int ambient,data1,data2,data3,data4,data5,data6,data7,data8;
int A,X;
char datax,dataA;

void tpa81(void)
{
    i2c_start();
    i2c_write(0xd0);
    i2c_write(1);
    i2c_start();
    i2c_write(0xd1);
    data[0]=i2c_read(1);
    data[1]=i2c_read(1);
    data[2]=i2c_read(1);
    data[3]=i2c_read(1);
    data[4]=i2c_read(1);
    data[5]=i2c_read(1);
    data[6]=i2c_read(1);
    data[7]=i2c_read(1);
    data[8]=i2c_read(0);
    i2c_stop();

    sprintf(text,"%2d %2d %2d %2d %2d %2d %2d %2d %2d",
            data[1],data[2],data[3],data[4],data[5],data[6],data[7],data[8],data[0]);

    dataA=data[0]; //ambient

    datax=data[1];
    if(datax<data[2])
    datax=data[2];
    if(datax<data[3])
    datax=data[3];
    if(datax<data[4])
    datax=data[4];
    if(datax<data[5])
    datax=data[5];
    if(datax<data[6])
    datax=data[6];
    if(datax<data[7])
    datax=data[7];
    if(datax<data[8])
    datax=data[8];

    return;
}

///////////////

////// kemiringan
int mir;
//////////
```

```
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
```

```
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: Off
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x47;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 691.200 kHz
// ADC Voltage Reference: AVCC pin
// ADC Auto Trigger Source: None
// Only the 8 most significant bits of
// the AD conversion result are used
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x84;

// I2C Bus initialization
i2c_init();

// LCD module initialization
lcd_init(16);

while (1)
      {
      // Place your code here

      ///////TPA81
      tpa81();
      /////////

     ////////ping
     ping();
     //////////////////

     /////////ir
     ir();
     ////////////////

     //////////kemiringan
     mir=read_adc(7);
     ////////////
```

```
    putchar('x');            /// pembuka

    if(spcA=='x')            /// ping dpn
    spcA=spcA-1;
    if(spcA=='y')
    spcA=spcA+1;
    putchar(spcA);

    if(spcL=='x')            /// ping kiri
    spcL=spcL-1;
    if(spcL=='y')
    spcL=spcL+1;
    putchar(spcL);

    if(spcR=='x')            /// ping kanan
    spcR=spcR-1;
    if(spcR=='y')
    spcR=spcR+1;
    putchar(spcR);

    putchar(flame);          /// uvtron

    if(mir=='x')             /// kemiringan
    mir=mir-1;
    if(mir=='y')
    mir=mir+1;
    putchar(mir);

    if(dr=='x')           /// ir kanan
    dr=dr-1;
    if(dr=='y')
    dr=dr+1;
    putchar(dr);

    if(dl=='x')           /// ir kiri
    dl=dl-1;
    if(dl=='y')
    dl=dl+1;
    putchar(dl);

     ////////// TPA81
    A=dataA;             /// ambient
    if(dataA=='x')
    A=A-1;
    if(dataA=='y')
    A=A+1;
    putchar(A);

    X=datax;          /// data panas
    if(datax=='x')
    X=X-1;
    if(datax=='y')
    X=X+1;
    putchar(X);
    ////////////////////

    putchar('y');          /// penutup


    delay_ms(75);

    };
}
```

MIKROKONTROLER AKTUATOR

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.3 Professional
Automatic Program Generator
© Copyright 1998-2007 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project : Terima ku
Version : 1
Date    : 2/12/2009
Author  : Ratana Chanda
Company : UKM
Comments:

Chip type           : ATmega16
Program type        : Application
Clock frequency     : 11.059200 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 256
*****************************************************/

#include <mega16.h>
#include <stdio.h>
#include <delay.h>
#include <math.h>

char buff[20];
char buff2[7];
char ping_kiri,ping_depan,ping_kanan;
char ir_kanan,ir_kiri;
char uvtron;
char kompas;
char TPA_dataA,TPA_dataX;
char kemiringan;
char seperlimapuluh;
char jalan;
int i;
int z=1,a=0,lt=1, by=1;
int pingD,pingR,pingL,irR,irL,uvt,kemirA;
float kmps,kemir;
int TPAdataA,TPAdataX;

#define lurus 1
#define lurus_a 2
#define lurus_b 3
#define belok_kiri 4
#define belok_kanan 5
#define stop 6
#define ha_ri 7
#define ha_nan 8
#define kipas_on 9
#define kipas_off 10

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x1B ;PORTA
#endasm
#include <lcd.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
```

```
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
  {
  rx_buffer[rx_wr_index]=data;
  if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
  if (++rx_counter == RX_BUFFER_SIZE)
    {
    rx_counter=0;
    rx_buffer_overflow=1;
    };
  };
   i++;
  buff[i]=data;

            if((buff[i]=='y')&&(buff[1]=='x'))
     {
     ping_depan=buff[2];
     ping_kiri=buff[3];
     ping_kanan=buff[4];
     uvtron=buff[5];
     kemiringan=buff[6];
     ir_kanan=buff[7];
     ir_kiri=buff[8];
     TPA_dataA=buff[9];
     TPA_dataX=buff[10];

      i=0;
     }

  if(buff[i]=='y')
  {
  i=0;
  }

    pingD=ping_depan;
    pingR=ping_kanan;
    pingL=ping_kiri;
    irR=ir_kanan;
    irL=ir_kiri;
    uvt=uvtron;
```

```
    kemirA=kemiringan;
    kemir=kemirA;
    kemir=((kemir-131)/255)*360;

    TPAdataA=TPA_dataA;
    TPAdataX=TPA_dataX;

}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

// Timer 0 output compare interrupt service routine
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
// Place your code here

seperlimapuluh++;
if(seperlimapuluh==25)
{
seperlimapuluh=0;

////////// cek lantai
if(kemir>10)
  {
   if ( (lt==1) || (lt==2) )
       lt=2;

  }

if(kemir<-10)
  {
   if ( (lt==1) || (lt==2) )
       lt=1;

  }

if((kemir>=-10) && (kemir<=10))
  {
   if ( (lt==1) )
        lt=1;
  if ( (lt==2) )
        lt=2;

  }

//////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// lt 1
if (lt==1)

{
// a=0 --> tidak cek api pada pertigaan
// a=1 --> cek api (kiri dan kanan) pada pertigaan

////////////////////////////////////////////////////////////////////// tidak terdeteksi api
if (uvt==0)
      {
       jalan=kipas_off;
      if (pingD>18)
       {
            if( ((pingL<50) && (pingR<60)) ) // jalan lurus antara 2 dinding
                  {
                  if ((( (pingL<19) && (pingL>=17) ) && ( (pingR<19) && (pingR>=17)) && ((irL>25 && irR>25)) ) )
                     jalan=lurus;
                  if (((pingL>=17) && (pingR<19)) || ((irL>25) && (irR<=25)) ) //koreksi kiri
                      jalan=lurus_a;
                  if (((pingL<19) && (pingR>=17)) || ((irL<=25) && (irR>25)) ) // koreksi kanan
                      jalan=lurus_b;
                  }


             if ( ( ((pingL>=61) && (irL>=29)) || ((pingR>=61) && (irR>=29)) ) && (pingD>=60) )
             {
                  if (((pingL>60) && (pingR<=32)) && ((irL>29) && (irR<=25)) )
                        // pertigaan----dpan jauh, kanan dkt, kiri  jauh
                   {
                        a=2;
                   }
                  if (((pingL<=32) && (pingR>60)) && ((irL<=25) && (irR>29)) )
                        // pertigaan----dpan jauh, kanan jauh, kiri dkt
                  {
                        a=3;
                  }

              }

       }

       if (pingD<=18)
       {
            if((pingL<50) && (pingR<50)) // jalan buntu
                  {
                  if (pingL>=20)
                   jalan=ha_nan;
                   if (pingR>=20)
                   jalan=ha_ri;
                  }

            if((pingL<50) && (pingR>50))  // tikungan kanan, hadap kanan
                  {
                  jalan=belok_kanan;
                  }

            if((pingL>50) && (pingR<50))  // tikungan kiri, hadap kiri
                  {
                  jalan=belok_kiri;
                  }

             if((pingL>40) && (pingR>40))  // per-tiga-an, cek api
                  {
                   a=1;
                  }

       }
```

```
 }

////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////// terdeteksi adanya api
if (uvt==1)
{
  if (TPA_dataX<=90)
  {
     jalan=kipas_off;
    if (pingD>19)
    {
    if ( (pingR<21) && (pingR>=19) )
      jalan=lurus;
    if ( (pingR<19) || (irR<=20) ) //koreksi kiri
      jalan=lurus_a;
    if ((pingR>=21) || (irR>20) ) // koreksi kanan
      jalan=lurus_b;
    if (  (pingL>20) && (pingR>21) )
     jalan=ha_nan;
   }

    if ((pingD<=19) && (pingR<21))
      {
      jalan=ha_ri;
      }

  }

  if (((pingD<=22) && ((TPA_dataX>90) && (TPA_dataX<=150))) || ((TPA_dataX>90) && (TPA_dataX<=150))  )
      {
      jalan=kipas_on;
      }

}

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////
}

if (lt==2)
{
////////////////////////////////////////////////// sisir kanan ////////////////////////////////////////lt 2
       if (TPA_dataX==36)
          by=0;  // bayi ditemukan

   if (pingD>19)
   {
    if ( (pingR<22) && (pingR>=20) )
      jalan=lurus;
   if ( (pingR<18) || (irR<=20) ) //koreksi kiri
      jalan=lurus_a;
   if ((pingR>=22) || (irR>20) ) // koreksi kanan
      jalan=lurus_b;
   if (  (pingL>20) && (pingR>22) )
     jalan=ha_nan;
   }

   if ((pingD<=19) && (pingR<22))
   {
            jalan=ha_ri;
   }


////////////////////////////////////////////////////////////////////////////////////////////////
}
```

```
}

}

// Declare your global variables here

void maju_full(unsigned int x,unsigned int y) // nanjak dan maju biasa
{
OCR1A=x;
OCR1B=y;
 //PORTD.4=OCR1A;
 PORTB.0=1;
 PORTB.1=0;

 //PORTD.5=OCR1B;
 PORTB.2=0;
 PORTB.3=1;

return;
}

void putar_kiri(int x,int y)
{
OCR1A=x;
OCR1B=y;

 //PORTD.4=OCR1A;
 PORTB.0=1;
 PORTB.1=0;

 //PORTD.5=OCR1B;
 PORTB.2=1;
 PORTB.3=0;

return;
}

void putar_kanan(int x, int y)
{
OCR1A=x;
OCR1B=y;

 //PORTD.4=OCR1A;
 PORTB.0=0;
 PORTB.1=1;

 //PORTD.5=OCR1B;
 PORTB.2=0;
 PORTB.3=1;

return;
}

void berhenti(void)
{
 PORTD.4=0;
 PORTB.0=1;
 PORTB.1=1;

 PORTD.5=0;
 PORTB.2=1;
 PORTB.3=1;

 return;
}
```

```
void kipas_putar(void)
{
PORTC.7=1;
PORTC.6=1;
PORTC.5=0;
return;
}

void kipas_mati(void)
{
PORTC.7=1;
PORTC.6=1;
PORTC.5=1;
return;
}

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=Out Func1=Out Func0=Out
// State7=T State6=T State5=T State4=T State3=0 State2=0 State1=0 State0=0
PORTB=0x00;
DDRB=0x0F;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=0 State5=0 State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0xE0;

// Port D initialization
// Func7=In Func6=In Func5=Out Func4=Out Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=0 State4=0 State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x30;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 10.800 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x05;
TCNT0=0x00;
OCR0=0xD7;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 43.200 kHz
// Mode: Ph. correct PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0xA1;
```

```
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x02;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: Off
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x90;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x47;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// LCD module initialization
lcd_init(16);

// Global enable interrupts
#asm("sei")

while (1)
    {
    // Place your code here

       lcd_clear();
       sprintf(buff2,"%3d ",pingL);
       lcd_puts(buff2);
       sprintf(buff2,"%3d ",pingD);
       lcd_puts(buff2);
       sprintf(buff2,"%3d ",pingR);
       lcd_puts(buff2);
       sprintf(buff2,"%1d ",lt);
       lcd_puts(buff2);
```

```
    sprintf(buff2,"%1d ",uvt);
    lcd_puts(buff2);
    lcd_gotoxy(0,1);
    sprintf(buff2,"%3d ",irL);
    lcd_puts(buff2);
    sprintf(buff2,"%3d ",TPA_dataX);
    lcd_puts(buff2);
    sprintf(buff2,"%3d ",irR);
    lcd_puts(buff2);
     sprintf(buff2,"%1d ",by);
    lcd_puts(buff2);

switch(a) /// pertigaan atau bukan
  {
       case 1:
       {
        putar_kiri(200,100);
        delay_ms(1700);
        if (uvt==1)
               {
               maju_full(200,100);
               delay_ms(1700);
                a=0;
               }
        if (uvt==0)
              {
              putar_kanan(200,100);
              delay_ms(2500);
              maju_full(200,100);
              delay_ms(1500);
               a=0;
               }

       }
       break;

       case 2:
       {
        putar_kiri(200,100);
        delay_ms(1700);
        if (uvt==1)
               {
               maju_full(200,100);
               delay_ms(2000);
                a=0;
               }
        if (uvt==0)
              {
              putar_kanan(200,100);
              delay_ms(1500);
              maju_full(200,100);
               a=0;
               }

       }
       break;

       case 3:
       {
        putar_kanan(200,100);
        delay_ms(1700);
        if (uvt==1)
               {
               maju_full(200,100);
               delay_ms(2000);
                a=0;
               }
```

```
 if (uvt==0)
       {
       putar_kiri(200,100);
       delay_ms(1500);
       maju_full(200,100);
        a=0;
        }

}
break;

case 0:
{


 switch(jalan)
 {
        case lurus:
        {
        maju_full(200,100);
        kipas_mati();
        }
        break;

        case lurus_a:
        {
        maju_full(250,50);
        kipas_mati();
        }
        break;
        case lurus_b:
        {
        maju_full(150,130);
        kipas_mati();
        }
        break;

        case ha_nan:
        {
        kipas_mati();
        putar_kanan(200,100);
       }
        break;

        case ha_ri:
        {
        kipas_mati();
        putar_kiri(200,100);
        }
        break;

         case belok_kanan:
        {
        kipas_mati();
        putar_kanan(200,100);
        delay_ms(900);
        maju_full(200,100);
        delay_ms(1700);
        }
        break;

         case belok_kiri:
        {
        kipas_mati();
        putar_kiri(200,100);
        delay_ms(900);
        maju_full(200,100);
```

```
                delay_ms(1700);
                }
                break;

                case kipas_on:
                {
                berhenti();
                kipas_putar();
                }
                break;

                case kipas_off:
                {
                kipas_mati();
                }
                break;
                case stop:
                {
                berhenti();
                }
                break;
            }

          }
          break;

      }

z++;
    };
}
```

SUBPROGRAM PENGGUNAAN INFRAMERAH (GP2D12)

```
/////// for 8 bit
int sr,sl,dr,dl;
char textr[16];
char textl[16];

void ir(void)
{
/// ir_kanan
sr=read_adc(0);
 dr=2141.72055 * (pow(sr,-1.078867)); /// sudah dlm cm

/// ir_kiri
sl=read_adc(1);
 dl=2141.72055 * (pow(sl,-1.078867)); /// sudah dlm cm

return;
}
```

SUBPROGRAM PENGGUNAAN ULTRASONIK (PING)

```
unsigned int countA=0;
unsigned int countL=0;
unsigned int countR=0;
float jarakA,jarakR,jarakL;
char spcA;
char spcL;
char spcR;
char PingA[16];
char PingL[16];
char PingR[16];

void ping(void)
            {
//========================== pingA / PING tengah
    countA=0;
    DDRA.4=1;
    PORTA.4=1;
    delay_us(5);
    PORTA.4=0;

    DDRA.4=0;
    PORTA.4=1;

    while (PINA.4==0) {};
    while (PINA.4==1)
     { countA++;
     }
    jarakA=((float)countA)/(242*3)*10;
    spcA=jarakA;
    sprintf(PingA,"%3.0f ",jarakA);

//========================== pingL / PING kiri
     countL=0;
    DDRA.5=1;
    PORTA.5=1;
    delay_us(5);
    PORTA.5=0;

    DDRA.5=0;
    PORTA.5=1;

    while (PINA.5==0) {};
    while (PINA.5==1)
```

```
      { countL++;
      }

    jarakL=((float)countL)/(242*3)*10;
    spcL=jarakL;
    sprintf(PingL,"%3.0f ",jarakL);

 //========================== pingR / PING kanan
     countR=0;
    DDRA.6=1;
    PORTA.6=1;
    delay_us(5);
    PORTA.6=0;

    DDRA.6=0;
    PORTA.6=1;

    while (PINA.6==0) {};
    while (PINA.6==1)
     { countR++;
     }

    jarakR=((float)countR)/(242*3)*10;
    spcR=jarakR;
    sprintf(PingR,"%3.0f ",jarakR);

return;
}
```

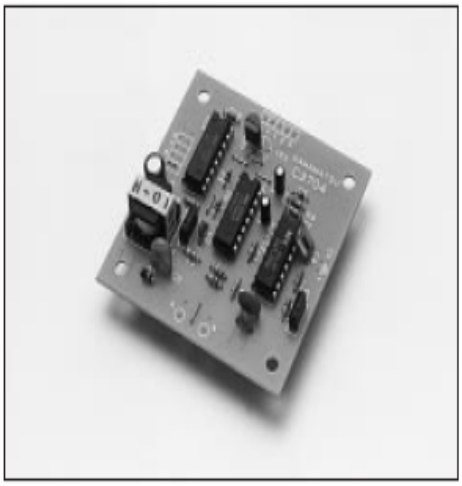
LAMPIRAN C
DATASHEET

Compact, Lightweight, Low Current Consumption, Low Cost Operates as High Sensitivity UV Sensor with UV TRON Suitable for Flame Detectors and Fire Alarms

Hamamatsu C3704 series UV TRON driving circuits are low current consuming, signal processing circuits for the UV TRON, well known as a high sensitivity ultraviolet detecting tube. The C3704 series can be operated as a UV sensor by connecting the UV TRON and applying DC low voltage, as they have both a high-voltage power supply and a signal processing circuit on the same printed circuit board.

Since background discharges of the UV TRON caused by natural excitation lights (such as a cosmic ray, scattered sunlight, etc.) can be cancelled in the signal processing circuit, the output signals from the C3704 series can be used without errors.

When the high sensitivity sensor "UV TRON R2868" (sold separately) is used, the flame from a cigarette lighter (flame length: 25mm) can be detected even from a distance of more than 5m.



APPLICATIONS

- Flame detectors for gas and oil lighters
- Fire alarms
- Combustion monitors for burners
- Electric spark detector
- UV photoelectric counter

SPECIFICATIONS

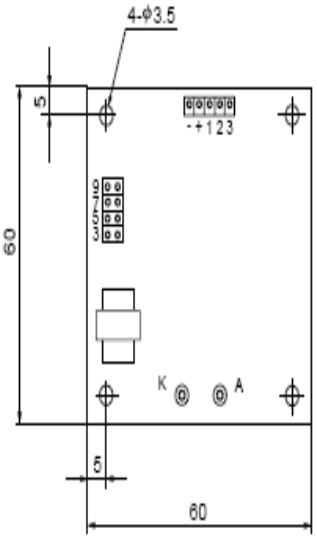
Dimensional outline ..... Figure 1
Weight ..... Approx. 20g
Output signal ..... Open collector Output (50 V, 100 mA Max.)
10 ms width pulse output (Note : 1)
UV TRON supply voltage ..... DC 350 V (Note : 2)
Quenching time ..... Approx. 50 ms
Operating temperature ..... -10 to +50°C
(with no condensation)
Suitable UV TRON ..... Low voltage operation UV TRON
(such as R2868)

	C3704	C3704-02	C3704-03
Input Voltage	10 to 30 Vdc	5Vdc ± 5%	6 to 9 Vdc
Current consumption	3 mA Max.	300µA Max.	300µA Max.

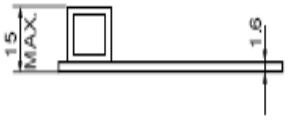
Note 1: The output pulse width can be extended up to about 100s by adding a capacitor to the circuit board.

Note 2: Since the output impedance of this power supply is extremely high, an ordinary voltmeter cannot be used. Use a voltmeter that has an input impedance of more than 10 GΩ.

Figure 1: Dimensional Outline (Unit : mm)



<Top View>



<Side View>

TPTA0034EA

Figure 2: Angular Sensitivity (Directivity)

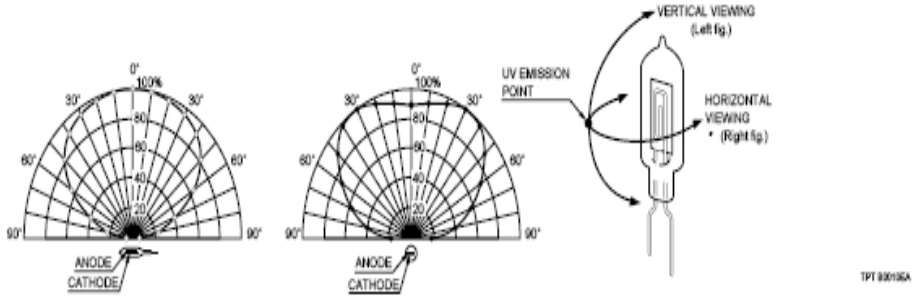


Figure 3: Dimensional Outline (Unit: mm)

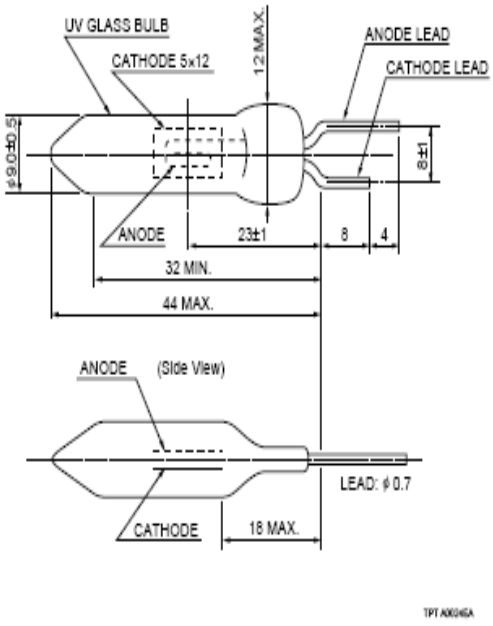
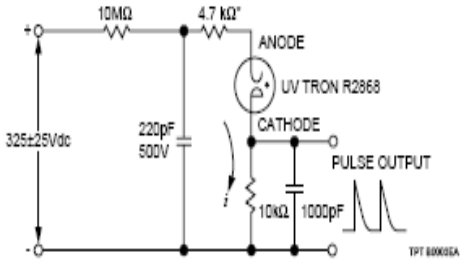
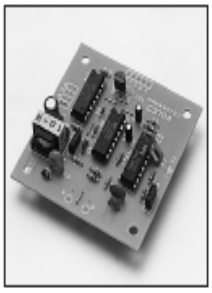


Figure 4: Recommended Operating Circuit



* Be sure to connect the 4.7 kΩ resistor within 2.5 cm from the anode lead end of UV TRON.

- UV TRON Driving Circuit C3704 series (Option)



Hamamatsu also provide the driving circuit C3704 series for R2868 operation.
C3704 series include a high voltage power supply and a signal processing circuit in printed circuit board, which allows to operate R2868 easily as a flame sensor with the low input voltage (DC 6 to 30 V) only.
For the details, please refer to the datasheet of C3704 series.

PING)))™ Ultrasonic Distance Sensor (#28015)

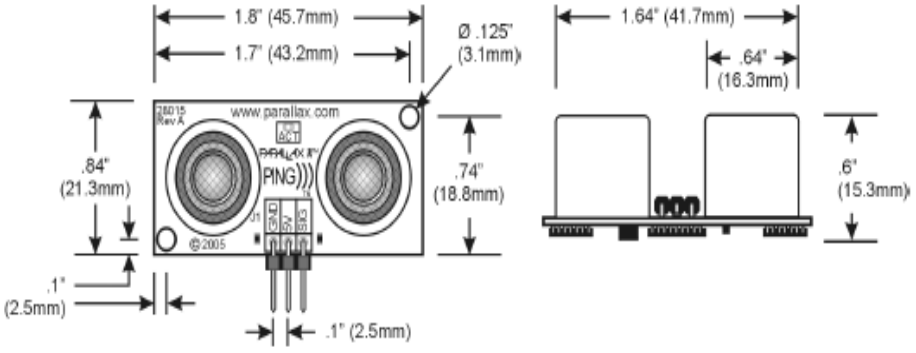
The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamp® or Javelin Stamp modules, SX or Propeller chips, or other microcontrollers, requiring only one I/O pin.

The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be calculated.

Features

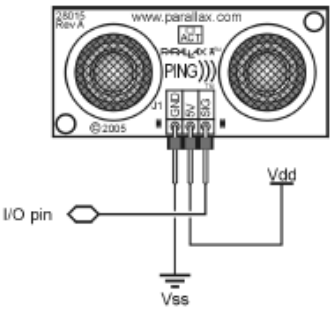
- Supply voltage – +5 VDC
- Supply current – 30 mA typ; 35 mA max
- Range – 2 cm to 3 m (0.8 inches to 3.3 yards)
- Input trigger – positive TTL pulse, 2 µs min, 5 µs typ.
- Echo pulse – positive TTL pulse, 115 µs to 18.5 ms
- Echo hold-off – 750 µs from fall of trigger pulse
- Burst frequency – 40 kHz for 200 µs
- Burst indicator LED shows sensor activity
- Delay before next measurement – 200 µs
- Size – 22 mm H x 46 mm W x 16 mm D (0.84 in x 1.8 in x 0.6 in)
- Operating temperature: 0 – 70° C.

Dimensions



Pin Definitions

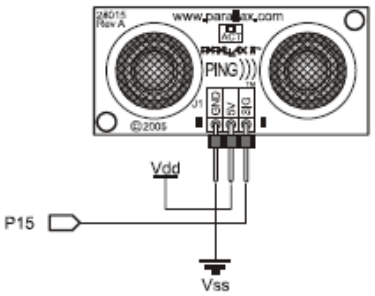
GND	Ground (Vss)
5 V	5 VDC (Vdd)
SIG	Signal (I/O pin)

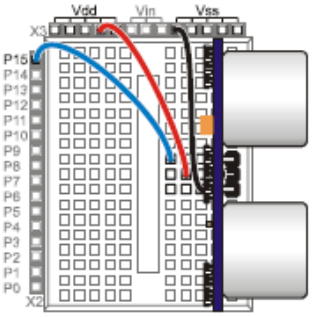


The PING))) sensor has a male 3-pin header used to supply ground, power (5 VDC) and signal. The header allows the sensor to be plugged into a solderless breadboard, or to be located remotely through the use of a standard servo extender cable (Parallax part #805-00002). Standard connections are shown in the diagram to the right.

Quick-Start Circuit

This circuit allows you to quickly connect your PING))) sensor to a BASIC Stamp® 2 via the Board of Education® breadboard area. The PING))) module's GND pin connects to Vss, the 5 V pin connects to Vdd, and the SIG pin connects to I/O pin P15. This circuit will work with the example BASIC Stamp program listed below.

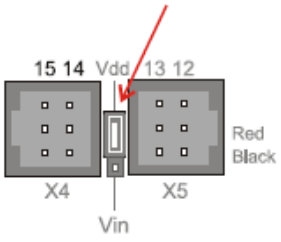




Extension Cable and Port Cautions

If you want to connect your PING))) sensor to a Board of Education platform using an extension cable, follow these steps:

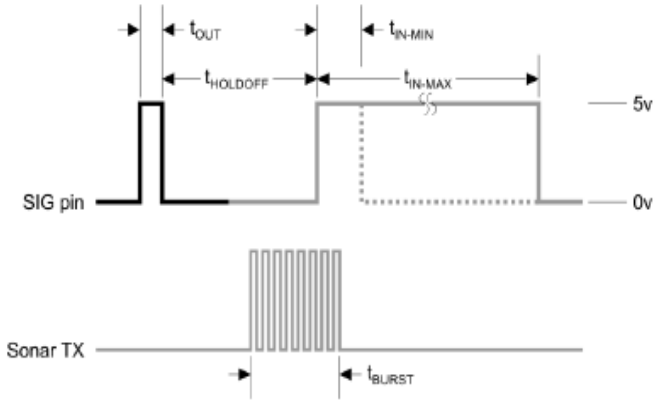
1. When plugging the cable onto the PING))) sensor, connect Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown. Then plug the cable into the port, matching the wire color to the labels next to the port.
4. If your Board of Education servo ports do not have a jumper, **do not use them with the PING))) sensor.** These ports only provide Vin, not Vdd, and this may damage your PING))) sensor. Go to the next step.
5. Connect the cable directly to the breadboard with a 3-pin header as shown above. Then, use jumper wires to connect Black to Vss, Red to Vdd, and White to I/O pin P15.



Board of Education Servo Port Jumper, Set to Vdd

Theory of Operation

The PING))) sensor detects objects by emitting a short ultrasonic burst and then "listening" for the echo. Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst. This burst travels through the air at about 1130 feet per second, hits an object and then bounces back to the sensor. The PING))) sensor provides an output pulse to the host that will terminate when the echo is detected, hence the width of this pulse corresponds to the distance to the target.

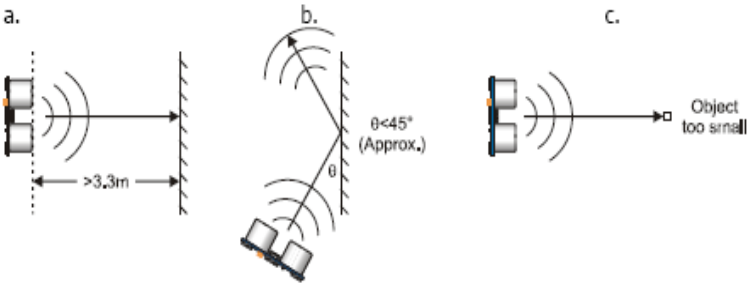


	Host Device	t_{OUT}	2 µs (min), 5 µs typical
	PING))) Sensor	$t_{HOLDOFF}$	750 µs
		t_{BURST}	200 µs @ 40 kHz
		t_{IN-MAX}	115 µs
		t_{IN-MIN}	18.5 µs

Practical Considerations for Use

Object Positioning

The PING))) sensor cannot accurately measure the distance to an object that: a) is more than 3 meters away, b) that has its reflective surface at a shallow angle so that sound will not be reflected back towards the sensor, or c) is too small to reflect enough sound back to the sensor. In addition, if your PING))) sensor is mounted low on your device, you may detect sound reflecting off of the floor.

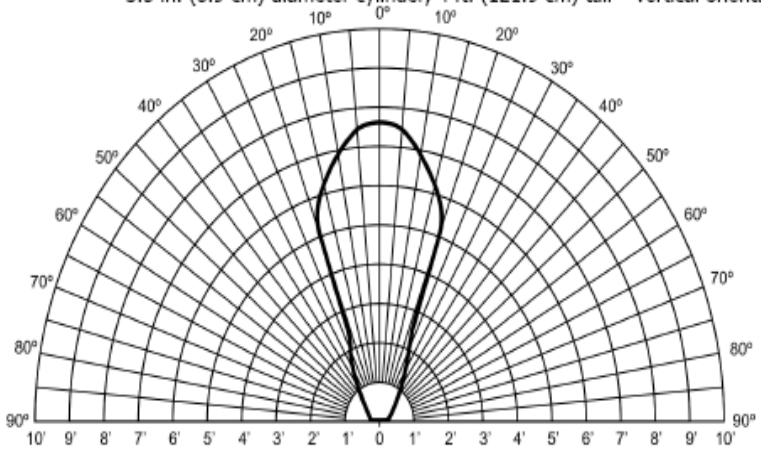


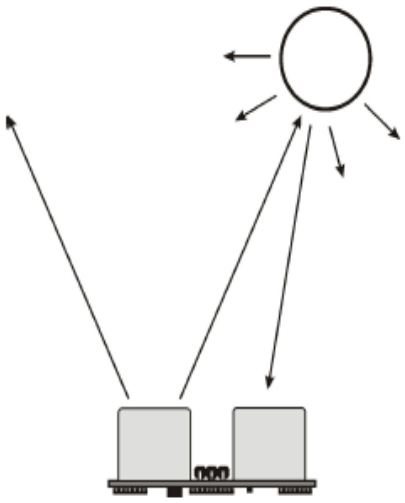
Test Data

The test data on the following pages is based on the PING))) sensor, tested in the Parallax lab, while connected to a BASIC Stamp microcontroller module. The test surface was a linoleum floor, so the sensor was elevated to minimize floor reflections in the data. All tests were conducted at room temperature, indoors, in a protected environment. The target was always centered at the same elevation as the PING))) sensor.

Test 1

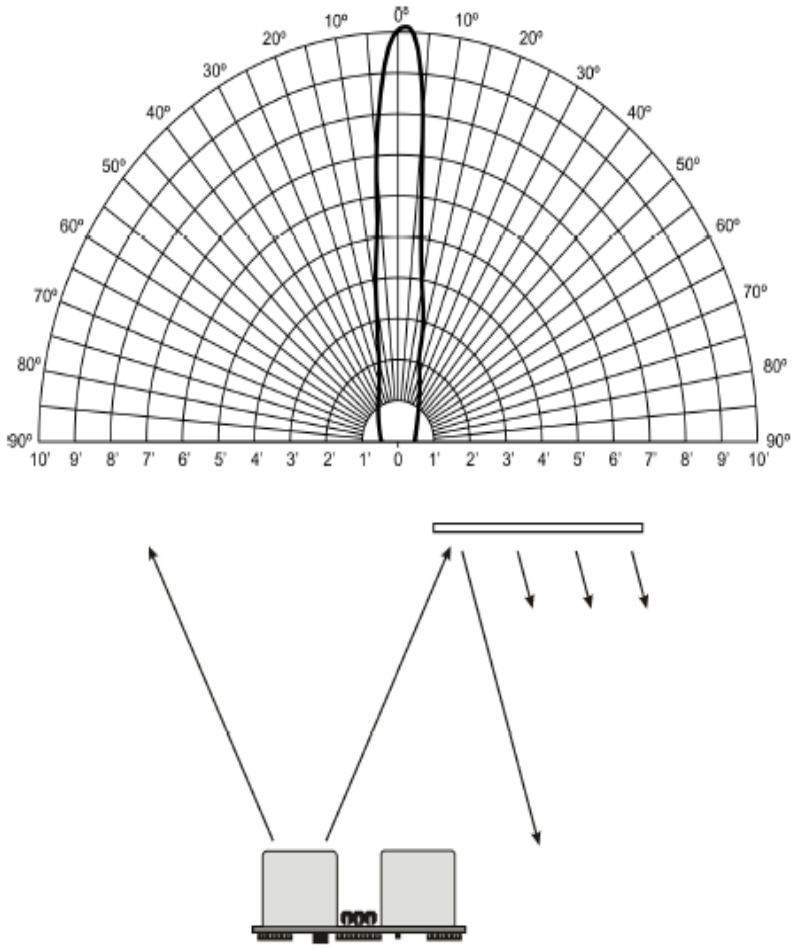
Sensor Elevation: 40 in. (101.6 cm)
Target: 3.5 in. (8.9 cm) diameter cylinder, 4 ft. (121.9 cm) tall – vertical orientation





Test 2

Sensor Elevation: 40 in. (101.6 cm)
Target: 12 in. x 12 in. (30.5 cm x 30.5 cm) cardboard, mounted on 1 in. (2.5 cm) pole
Target positioned parallel to backplane of sensor



TPA81 Thermopile Array

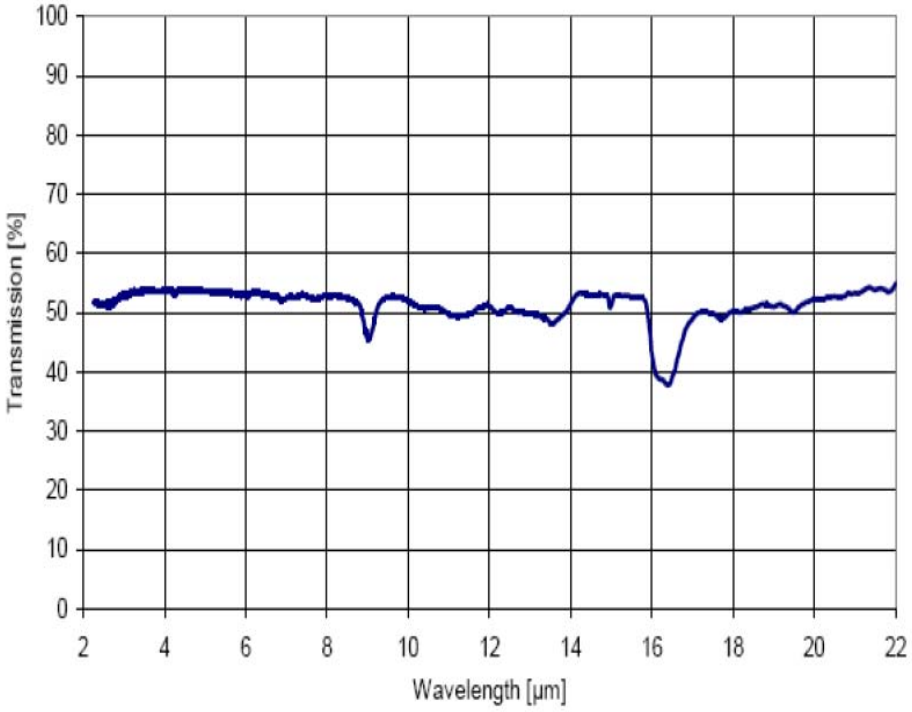
Technical Specification

Introduction

The TPA81 is a thermopile array detecting infra-red in the 2um-22um range. This is the wavelength of radiant heat. The Pyro-electric sensors that are used commonly in burglar alarms and to switch on outside lights, detect infra-red in the same waveband. These Pyro-electric sensors can only detect a change in heat levels though - hence they are movement detectors. Although useful in robotics, their applications are limited as they are unable to detect and measure the temperature of a static heat source. Another type of sensor is the thermopile array. These are used in non-contact infra-red thermometers. They have a very wide detection angle or field of view (FOV) of around 100° and need either shrouding or a lens or commonly both to get a more useful FOV of around 12°. Some have a built in lens. More recently sensors with an array of thermopiles, built in electronics and a silicon lens have become available. This is the type used in the TPA81. It has a array of eight thermopiles arranged in a row. The TPA81 can measure the temperature of 8 adjacent points simultaneously. The TPA81 can also control a servo to pan the module and build up a thermal image. The TPA81 can detect a candle flame at a range 2 metres (6ft) and is unaffected by ambient light!

Spectral Response

The response of the TPA81 is typically 2µm to 22µm and is shown below:



Field Of View

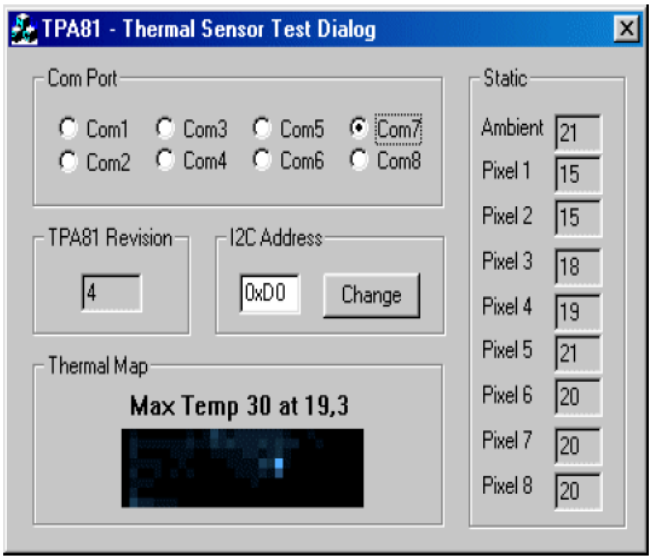
The typical field of view of the TPA81 is 41° by 6° making each of the eight pixels 5.12° by 6°. The array of eight pixels is orientated along the length of the PCB - that's from top to bottom in the diagram below. Pixel number one is nearest the tab on the sensor - or at the bottom in the diagram below.

Sensitivity

Here's some numbers from one of our test modules:
For a candle, the numbers for each of the eight pixels at a range of 1 meter in a cool room at 12°C are:
11 10 11 12 12 29 15 13 (All °C)
You can see the candle showing up as the 29°C reading. At a range of 2 meters this reduces to 20°C - still around 8° C above ambient and easily
detectable. At 0.6 meter (2ft) its around 64°C. At 0.3 meter (1ft) its 100°C+.

In a warmer room at 18°C, the candle measures 27°C at 2 meters. This is because the candle only occupies a small part of the sensors field of view and the candles point heat source is
added to the back ground ambient - not swamped by it. A human at 2 meters will show up as around 29°C with a background 20°C ambient.

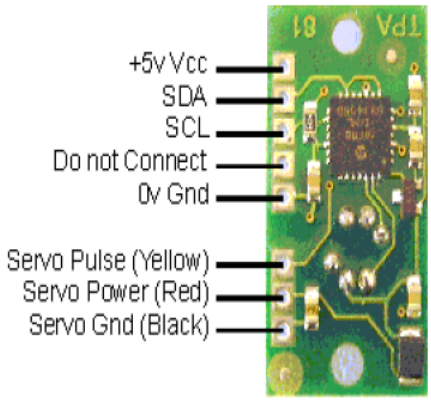
The following is a snapshot of our test program. It displays a 32x8 bitmap produced by using a servo to pan the sensor. If you want a copy of this windows based program, its here, but you will need an RF04/CM02 to connect the TPA81 to your PC. Here you can see a candle flame about a meter away showing up as the bright spot.



Connections

All communication with the TPA81 is via the I2C bus. If you are unfamiliar with the I2C bus, there is a tutorial which will help. The TPA81 uses our standard I2C 5 pin connection layout. The "Do Not Connect" pin should be left unconnected. It is actually the CPU MCLR line and is used once only in our workshop to program the PIC16F88 on-board after assembly, and has an internal pull-up resistor. The SCL and SDA lines should each have a pull-up resistor to +5v somewhere on the I2C bus. You only need one pair of resistors, not a pair for every module. They are normally located with the bus master rather than the slaves. The TPA81 is always a slave - never a bus master. If you need them, I recommend 1.8k resistors. Some modules such as the OOPic already have pull-up

resistors and you do not need to add any more. A servo port will connect directly to a standard RC servo and is powered from the modules 5v supply. We use an HS311. Commands can be sent to the TPA81 to position the servo, the servo pulses are generated by the TPA81 module.



Registers

The TPA81 appears as a set of 10 registers.

Register	Read	Write
0	Software Revision	Command Register
1	Ambient Temperature °C	Servo Range (V6 or higher only)
2	Pixel 1 Temperature °C	N/A
3	Pixel 2	N/A
4	Pixel 3	N/A
5	Pixel 4	N/A
6	Pixel 5	N/A
7	Pixel 6	N/A
8	Pixel 7	N/A
9	Pixel 8	N/A

Only registers 0, and 1 can be written to. Register 0 is the command register and is used to set the servo position and also when changing the TPA81's I2C address. It cannot be read. Reading from register 0 returns the TPA81 software revision. Writing to register 1 sets the servo range - see below. It cannot be read back, reading register 1 reads the ambient temperature.

There are 9 temperature readings available, all in degrees centigrade (°C). Register 1 is the ambient temperature as measured within the sensor. Registers 2-9 are the 8 pixel temperatures. Temperature acquisition is continuously performed and the readings will be correct approx 40mS after the sensor points to a new position.

Servo Position

Commands 0 to 31 set the servo position. There are 32 steps (0-31) which typically represent 180° rotation on a Hitec HS311 servo. The calculation is SERVO_POS*60+540uS. So the range of the servo pulse is 0.54mS to

2.4mS in 60uS steps. Writing any other value to the command register will stop the servo pulses.

Command		Action
Decimal	Hex	
0	0x00	Set servo position to minimum
nn	nn	Set servo position
31	0x1F	Set servo position to maximum
160	0xA0	1st in sequence to change I2C address
165	0xA5	3rd in sequence to change I2C address
170	0xAA	2nd in sequence to change I2C address

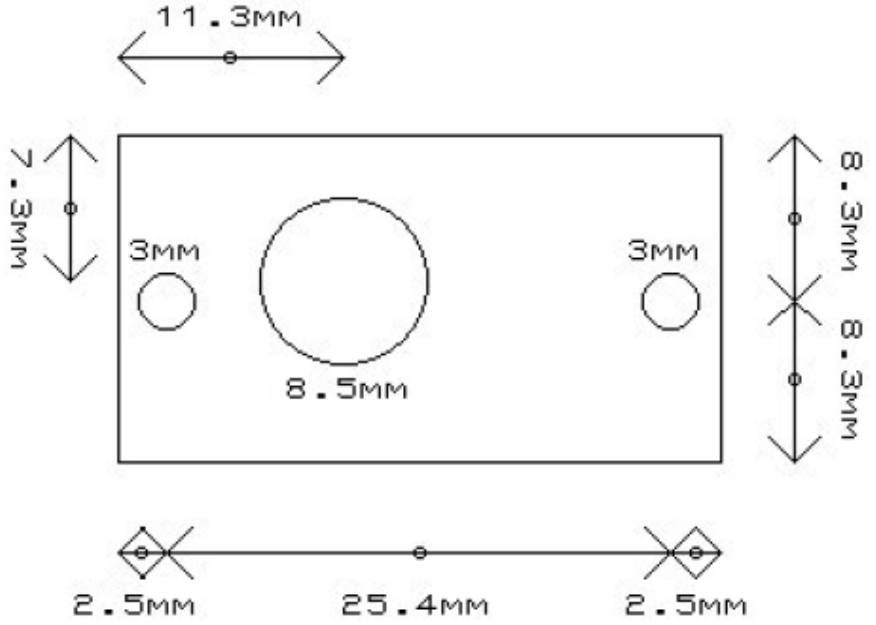
Firmware Version 6 or higher.

As from Version 6 (March 2005) we have added a new write only register (register 1) to allow you to vary the range of the servo stepping. It defaults to the same 180 degree range on a Hitec HS311 servo as earlier versions. You can write values from 20 to 120 to the range register. If you attempt to write a value less than 20, it will be set to 20. If you attempt to write a value greater than 120, it will be set to 120. The calculation for the range in uS is ((31*ServoRange)/2). Setting a range of 20 will give a range of (31*20)/2 or 310uS. Setting a range of 120 will give a range of (31*120)/2 or 1860uS. In all cases the available range is centered on the servo's mid position of 1500uS. So in the first example, the 310uS range will be from 1345uS to 1655uS (Or 1.345 to 1.655mS if you prefer). The second example of 1860uS range centered on 1500uS gives a range of 570uS to 2430uS. On power up the range register is set to 120, which give the same range as earlier versions.

Changing the I2C Bus Address

To change the I2C address of the TPA81 you must have only one module on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of a TPA81 currently at 0xD0 (the default shipped address) to 0xD2, write the following to address 0xD0; (0xA0, 0xAA, 0xA5, 0xD2). These commands must be sent in the correct sequence to change the I2C address, additionally, No other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 0, which means 4 separate write transactions on the I2C bus. Additionally, there MUST be a delay of at least 50uS between the writing of each byte of the address change sequence. When done, you should label the sensor with its address, if you lose track of the module addresses, the only way to find out what it is to search all the addresses one at a time and see which one responds. The TPA81 can be set to any of eight I2C addresses - 0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE. The factory default shipped address is 0xD0.

Mechanical Layout



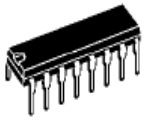
SGS-THOMSON MICROELECTRONICS

L293D
L293DD

PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES

- 600mA OUTPUT CURRENT CAPABILITY PER CHANNEL
- 1.2A PEAK OUTPUT CURRENT (non repetitive) PER CHANNEL
- ENABLE FACILITY
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)
- INTERNAL CLAMP DIODES

SO(12+4+4)



Powerdip (12+2+2)

ORDERING NUMBERS:

L293DD

L293D

DESCRIPTION

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors.

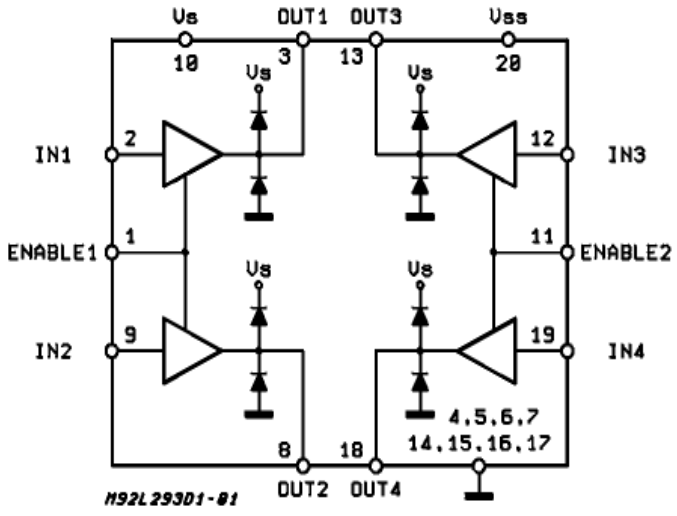
To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking

The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

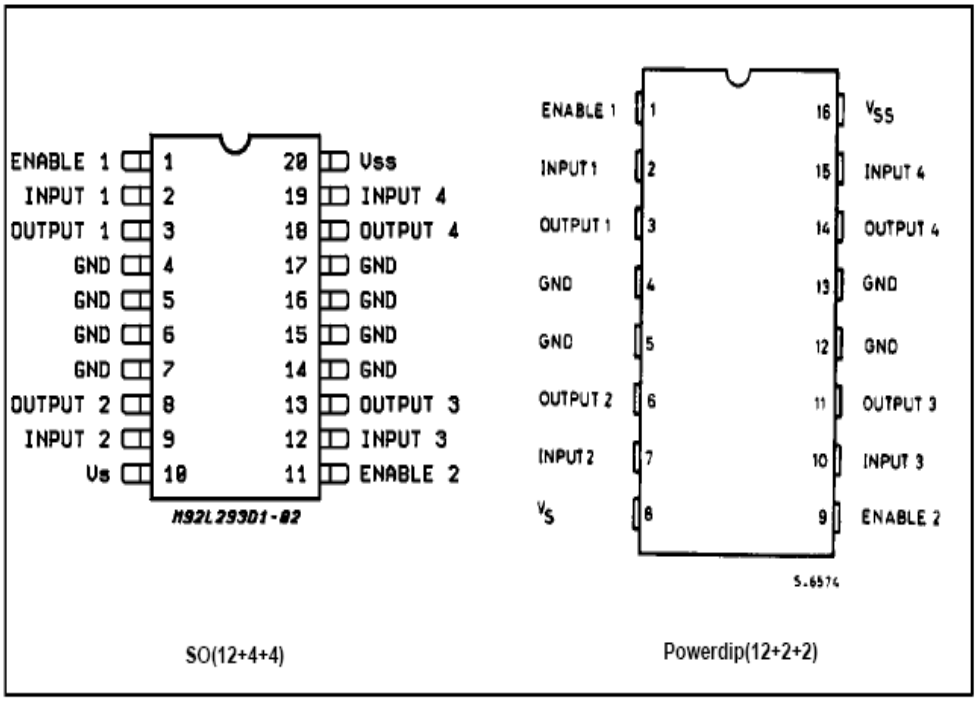
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Supply Voltage	36	V
V_{SS}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{en}	Enable Voltage	7	V
I_o	Peak Output Current (100 µs non repetitive)	1.2	A
P_{tot}	Total Power Dissipation at T_{pins} = 90 °C	4	W
T_{stg}, T_j	Storage and Junction Temperature	– 40 to 150	°C

PIN CONNECTIONS (Top view)



SO(12+4+4) Powerdip(12+2+2)

THERMAL DATA

Symbol	Decription		DIP	SO	Unit
$R_{th\ j\text{-}pins}$	Thermal Resistance Junction-pins	max.	–	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance junction-ambient	max.	80	50 (*)	°C/W
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	max.	14	–	

(*) With 6sq. cm on board heatsink.

ELECTRICAL CHARACTERISTICS (for each channel, V_S = 24 V, V_{SS} = 5 V, T_{amb} = 25 °C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 10)		V_{SS}		36	V
V_{SS}	Logic Supply Voltage (pin 20)		4.5		36	V
I_S	Total Quiescent Supply Current (pin 10)	V_i = L ; I_O = 0 ; V_{en} = H		2	6	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	24	mA
		V_{en} = L			4	mA
I_{SS}	Total Quiescent Logic Supply Current (pin 20)	V_i = L ; I_O = 0 ; V_{en} = H		44	60	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	22	mA
		V_{en} = L		16	24	mA
V_{IL}	Input Low Voltage (pin 2, 9, 12, 19)		– 0.3		1.5	V
V_{IH}	Input High Voltage (pin 2, 9, 12, 19)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
I_{IL}	Low Voltage Input Current (pin 2, 9, 12, 19)	V_{IL} = 1.5 V			– 10	µA
I_{IH}	High Voltage Input Current (pin 2, 9, 12, 19)	2.3 V ≤ V_{IH} ≤ V_{SS} – 0.6 V		30	100	µA
$V_{en\,L}$	Enable Low Voltage (pin 1, 11)		– 0.3		1.5	V
$V_{en\,H}$	Enable High Voltage (pin 1, 11)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
$I_{en\,L}$	Low Voltage Enable Current (pin 1, 11)	$V_{en\,L}$ = 1.5 V		– 30	– 100	µA
$I_{en\,H}$	High Voltage Enable Current (pin 1, 11)	2.3 V ≤ $V_{en\,H}$ ≤ V_{SS} – 0.6 V			± 10	µA
$V_{CE(sat)H}$	Source Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = – 0.6 A		1.4	1.8	V
$V_{CE(sat)L}$	Sink Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = + 0.6 A		1.2	1.8	V
V_F	Clamp Diode Forward Voltage	I_O = 600nA		1.3		V
t_r	Rise Time (*)	0.1 to 0.9 V_O		250		ns
t_f	Fall Time (*)	0.9 to 0.1 V_O		250		ns
t_{on}	Turn-on Delay (*)	0.5 V_i to 0.5 V_O		750		ns
t_{off}	Turn-off Delay (*)	0.5 V_i to 0.5 V_O		200		ns

(*) See fig. 1.

TRUTH TABLE (one channel)

Input	Enable (*)	Output
H	H	H
L	H	L
H	L	Z
L	L	Z

Z = High output impedance
(*) Relative to the considered channel

Figure 1: Switching Times

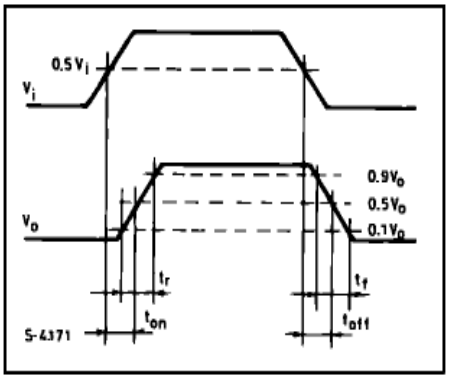
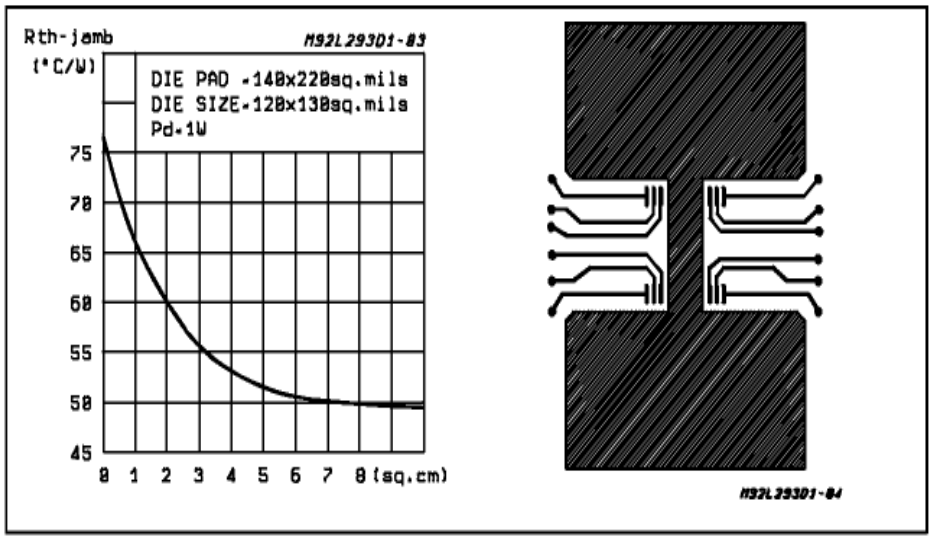
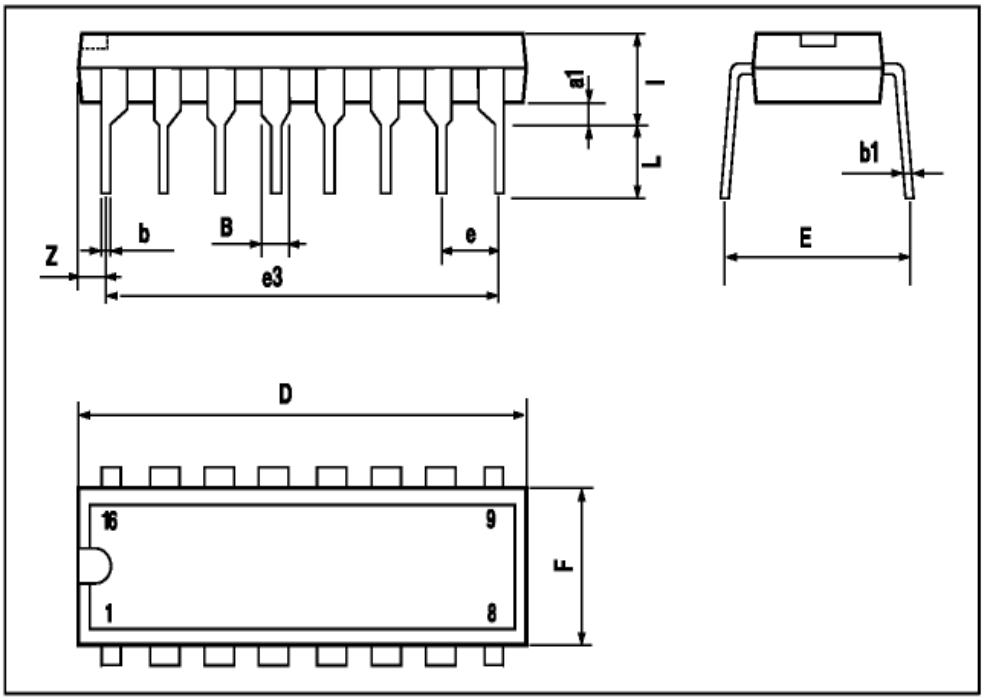


Figure 2: Junction to ambient thermal resistance vs. area on board heatsink (SO12+4+4 package)



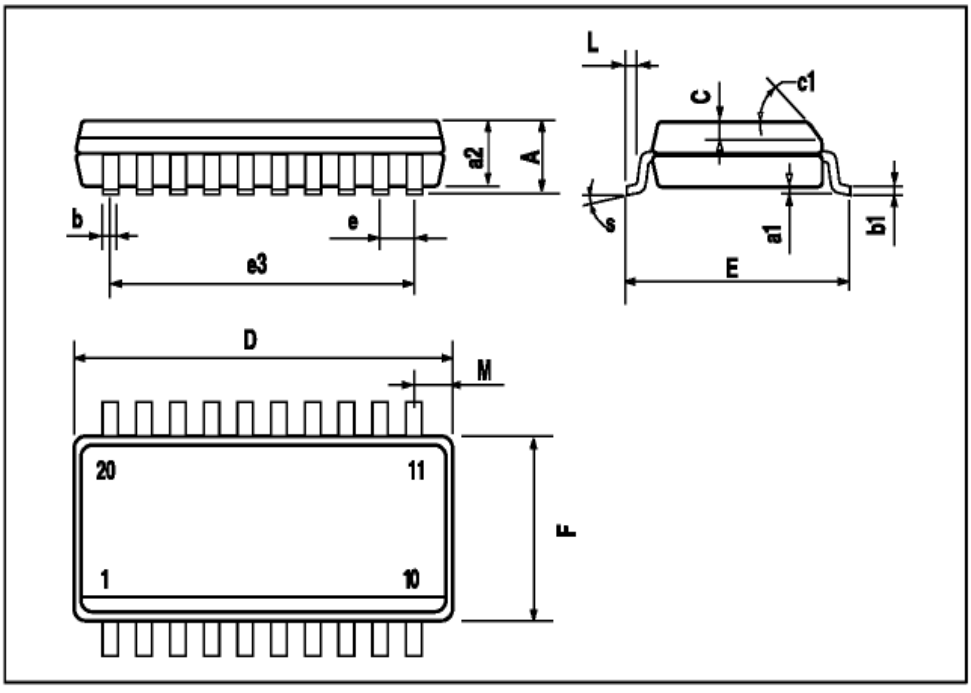
POWERDIP16 PACKAGE MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
a1	0.51			0.020		
B	0.85		1.40	0.033		0.055
b		0.50			0.020	
b1	0.38		0.50	0.015		0.020
D			20.0			0.787
E		8.80			0.346	
e		2.54			0.100	
e3		17.78			0.700	
F			7.10			0.280
I			5.10			0.201
L		3.30			0.130	
Z			1.27			0.050



SO20 PACKAGE MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			2.65			0.104
a1	0.1		0.2	0.004		0.008
a2			2.45			0.096
b	0.35		0.49	0.014		0.019
b1	0.23		0.32	0.009		0.013
C		0.5			0.020	
c1		45			1.772	
D		1	12.6		0.039	0.496
E	10		10.65	0.394		0.419
e		1.27			0.050	
e3		11.43			0.450	
F		1	7.4		0.039	0.291
G	8.8		9.15	0.346		0.360
L	0.5		1.27	0.020		0.050
M			0.75			0.030
S	8° (max.)					



Sharp GP2D12 Analog Distance Sensor (#605-00003)

General Description

The Sharp GP2D12 is an analog distance sensor that uses infrared to detect an object between 10 and 80 cm away. The GP2D12 provides a non-linear voltage output in relation to the distance an obj is from the sensor and interfaces easily using any analog to digital converter.

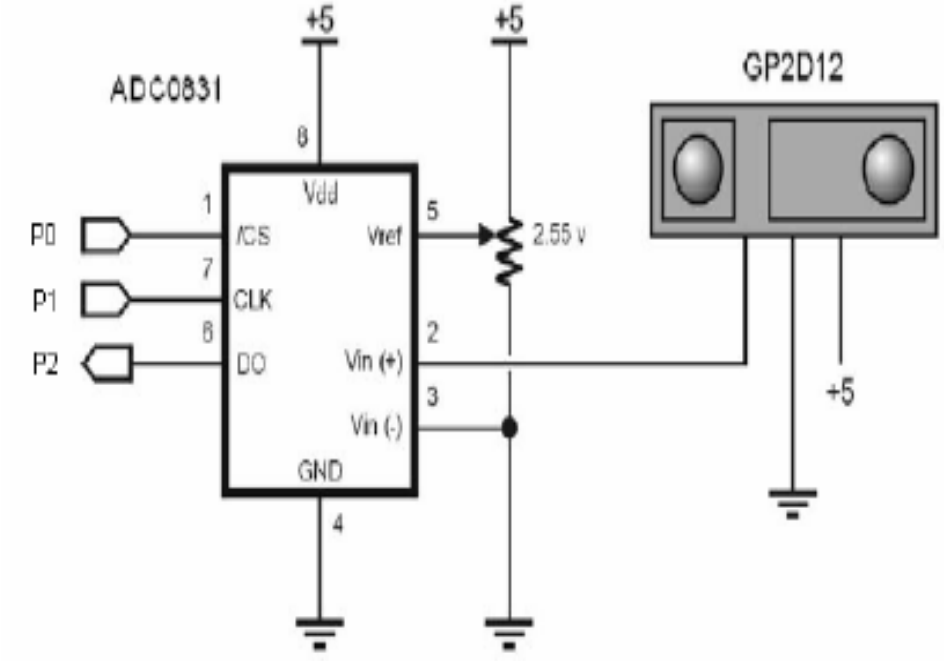
Features

- High immunity to ambient light and color of object
- No external control circuitry required
- Sensor includes convenient mounting holes
- Compatible with all BASIC Stamp® and SX microcontrollers

Application Ideas

- Robot range finder
- Halloween prop activation

Quick Start Circuit

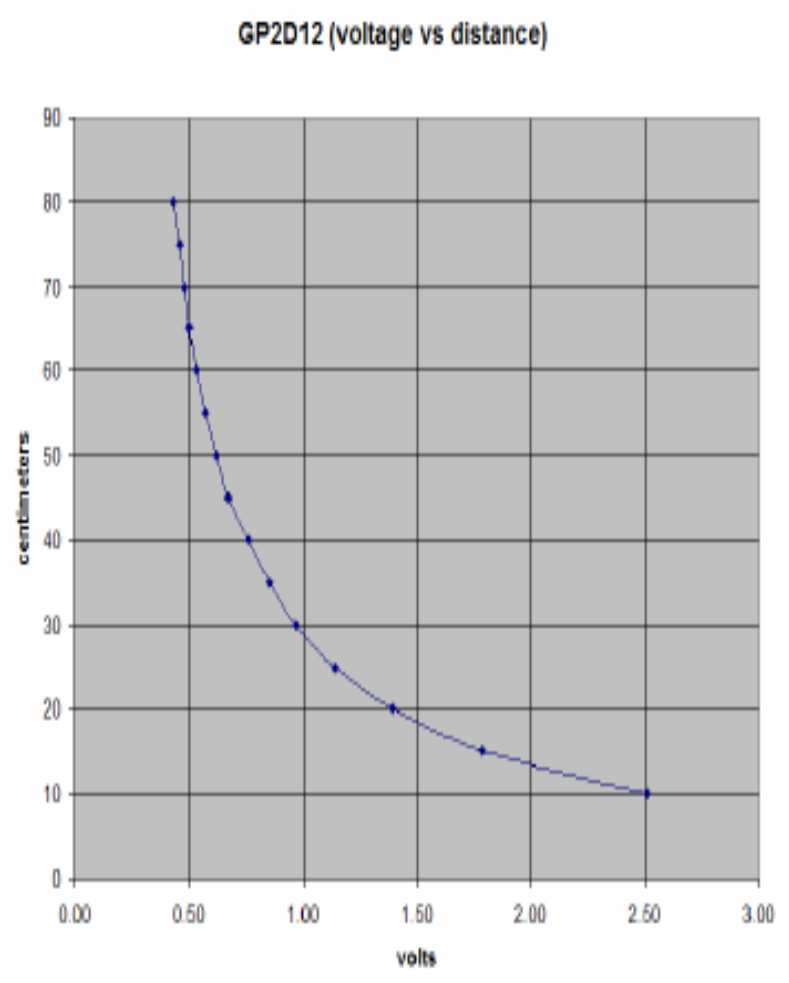


Connecting and Testing

Connect the GP2D12 to your analog to digital converter as shown in the circuit on the previous page. The potentiometer connected to the Vref pin on the ADC0831 is being used as a voltage divider to set the reference voltage to 2.55 volts. On the ADC0831 this will give a value of 0 to 255 for an input voltage of 0 to 2.55 volts. This gives us a resolution of 0.01 volts per step from the ADC. If you are using a different analog to digital converter, you may want to adjust the potentiometer to get the best results from your particular ADC.

Calibration

Because the output of the GP2D12 is not linear, we need a way to determine what distances correspond to what voltages. One way of calibrating your sensor is by measuring the voltage output of the GP2D12 at given fixed distances, in centimeters, as shown in the chart below. Once you have this information you can plug these numbers into the EEPROM DATA statements in the program. The table of data is used by a routine in the program to calculate the distances, which are then displayed on the Debug Terminal, along with the voltage output from the sensor.



Sensitivity

The usable range of the GP2D12 is between 10 cm and 80 cm. The readings for objects closer than 10 cm are unstable and therefore not usable.

Resources and Downloads

Check out the Sharp GP2D12 Analog Distance Sensor product page for example programs, the manufacturer datasheet and more:

http://www.parallax.com/detail.asp?product_id=605-00003

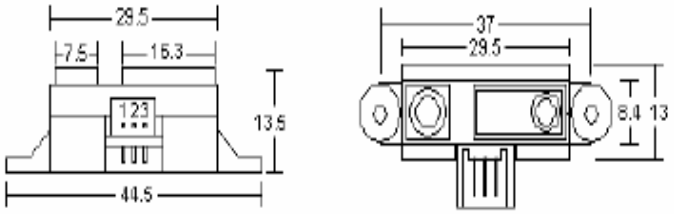
Specifications

Symbol	Quantity	Minimum	Typical	Maximum	Units
Vcc	Supply Voltage †	4.5	5.0	5.5	V
Topr	Operating Temperature †	-10	-	+60	°C
Tstg	Storage Temperature †	-40	-	+70	°C
ΔL	Distance Measuring Range †	10	-	80	cm
Vo	Output Terminal Voltage (L=80 cm) †	0.25	0.4	0.55	V
ΔVo	Output change at L=80 cm to 10 cm †	1.75	2.0	2.25	V
Icc	Average Dissipation Current (L=80cm) †	-	33	50	mA

† data obtained from Sharp's GP2D12 datasheet

Pin Definitions and Ratings

Pin	Name	Function
1	Vo	Voltage Output
2	GND	Ground
3	Vcc	Supply Voltage



*All units in millimeters (mm)