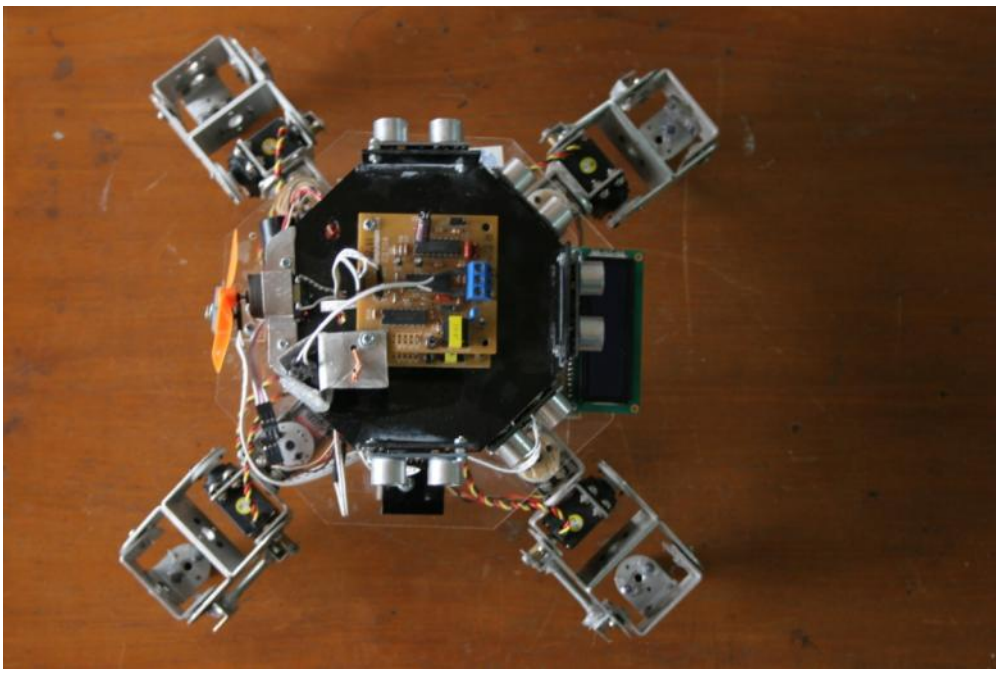
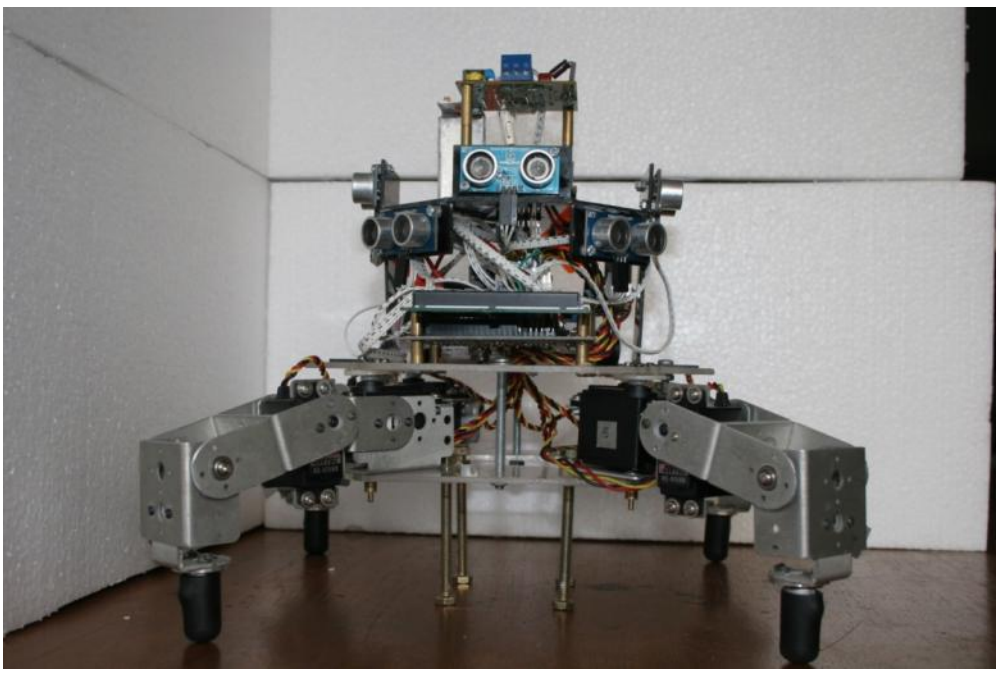
LAMPIRAN A

FOTO ROBOT *QUADRUPED*

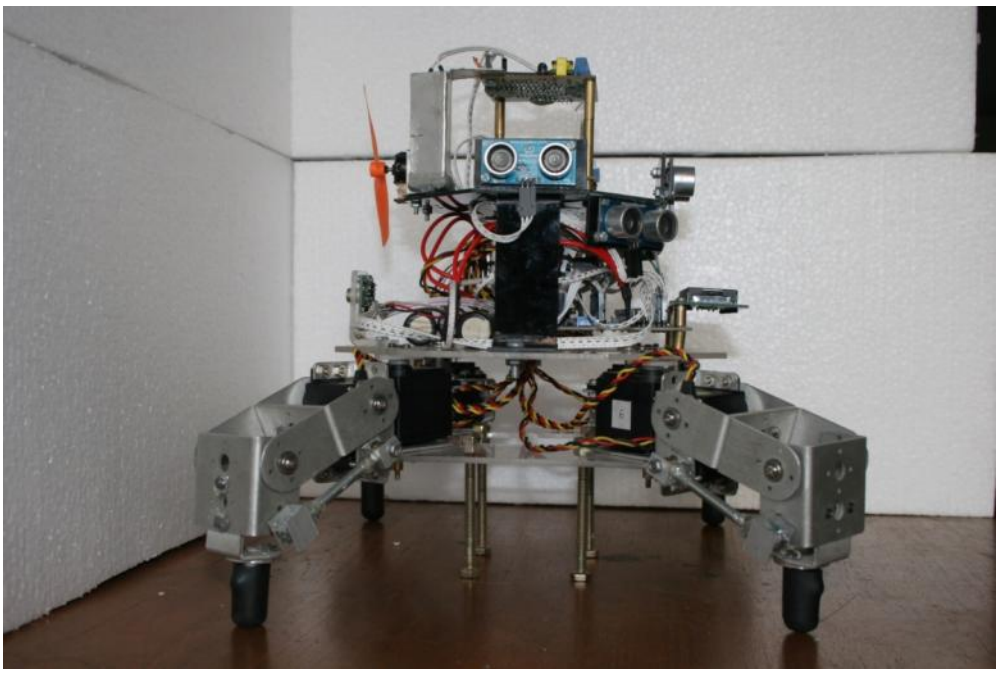
Tampak Atas



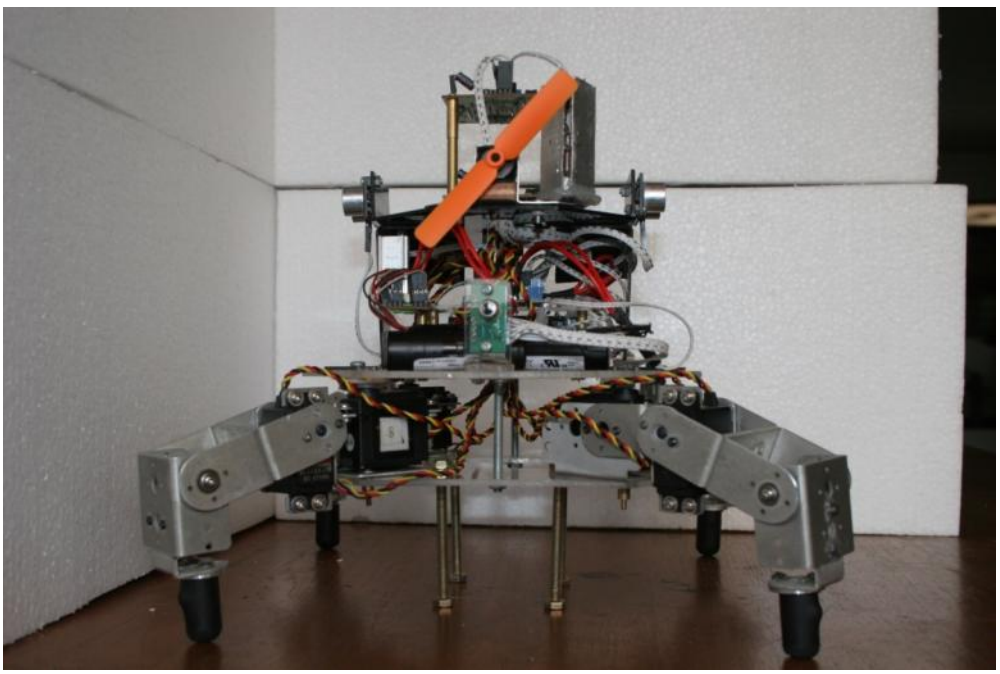
Tampak Depan



Tampak Samping



Tampak Belakang



LAMPIRAN B

PROGRAM PADA PENGONTROL MIKRO ATMEGA16 DAN ATTINY2313

PROGRAM UTAMA

PENGONTROL MIKRO ATMEGA16

```
/*****************************************************
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    : 11/18/2008
Author  : F4CG
Company : F4CG
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 <delay.h>
#include <stdio.h>
 unsigned int count=0;
 float jarak;
 int x, data,dataxy;
 char pilihan,status,jalan;

unsigned int a,b,c,d,e,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,z;
unsigned char text[16],text1[16];

// I2C Bus functions
#asm
   .equ __i2c_port=0x12 ;PORTD
   .equ __sda_bit=7
   .equ __scl_bit=6
#endasm
#include <i2c.h>


#asm
   .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

#include <ping.h>
```

```
// ------------------------------------------------------------------------------------------------------------
//                                        PROGRAM THERMAL ARRAY
// ------------------------------------------------------------------------------------------------------------

 void thermal(void)
       {
      i2c_start();
      i2c_write(0xD0);
      i2c_write(1);

      i2c_start();
      i2c_write(0xD1);
      m=i2c_read(1);
      n=i2c_read(1);
      o=i2c_read(1);
      p=i2c_read(1);
      q=i2c_read(1);
      r=i2c_read(1);
      s=i2c_read(1);
      u=i2c_read(1);
      v=i2c_read(0);
      i2c_stop();
      delay_ms(200);
      lcd_clear();
      sprintf(text,"%2u %2u %2u %2u %2u  %2u %2u %2u %2u",m,n,o,p,q,r,s,u,v);
      lcd_puts(text);
      delay_ms(500);
      }
// ------------------------------------------------------------------------------------------------------------
//                                        PROGRAM MAJU
// ------------------------------------------------------------------------------------------------------------
void go(void)
{
putchar('A');
      if(a>=17)
      {
      putchar('A');
        if (c<=11)
        {
          putchar('L');
          putchar('A');
        }
        else
        {
          if (d<c)
          {
          putchar('R');
          putchar('A');
          }
          else
          {
          putchar('L');
          putchar('L');
          putchar('L');
          putchar('A');
```

```
        }
       }

     }
     else
     {
      if(d<=30)
      {
        if (c>=d)
        {
        putchar('R');
        }
        else
        {
        putchar('L');
        }
      }
      else
      {
      putchar('L');
      }
     }
 }

// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
satu();
//dua();
tiga();
empat();
//lima();
lcd_gotoxy(0,0);
sprintf(text,"%3u %3u %3u %3u %3u",a,b,c,d,e);
lcd_puts(text);
//      a++;
//
//      if (PINA.1=1) status=2;
}



// Declare your global variables here

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=P State0=T
PORTA=0x00;
DDRA=0x00;
```

```
// Port B initialization
// Func7=In Func6=In Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=T State6=T State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTB=0x00;
DDRB=0x3F;


// 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: 10.800 kHz
// Mode: CTC top=OCR1A
// 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: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0D;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x1A;
OCR1AL=0x30;
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=0x10;

// I2C Bus initialization
i2c_init();

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x18;
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);

// Watchdog Timer initialization
// Watchdog Timer Prescaler: OSC/2048k
#pragma optsize-
WDTCR=0x1F;
WDTCR=0x0F;
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

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


z==0;
while (1)
```

```
    {

    // Place your code here
while (PINA.1==0)
  {
  go();
  lcd_clear();
  lcd_putsf("GA ADA API");
  delay_ms(500);
  }
while (PORTA.1=1)
  {
  putchar('A');
  delay_ms(3000);
  while (PINB.7==0)
  {
  lcd_clear();
  lcd_putsf("MANA APINYA???");
  delay_ms(500);
  putchar('T');
  delay_ms(250);
  putchar('L');
  }

  while (PORTB.7=1)
  {
  putchar('L');
  putchar('L');
  putchar('C');
  thermal();
  if (n>75||o>75||p>75||q>75||r>75||s>75||u>75||v>75)
  {
  putchar('T');
  delay_ms(500);
  lcd_clear();
  lcd_putsf("API");
  delay_ms(500);
  PORTA.0=1;
  delay_ms(3000);
  PORTA.0=0;
  delay_ms(3000);

 }
 }
 }
 };
 }
```

PENGONTROL MIKRO ATTINY2313

```
/*****************************************************
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    : 12/6/2008
Author  : Willisun Tanu
Company : KRCI
Comments:


Chip type           : ATtiny2313
Clock frequency     : 8.000000 MHz
Memory model        : Tiny
External SRAM size  : 0
Data Stack size     : 32
*****************************************************/

#include <tiny2313.h>
#include <stdio.h>
#include <delay.h>

char posisi_1(void);
char posisi_2(void);
char posisi_3(void);
char posisi_4(void);
char diem(void);
char muter_L(void);
char muter_R(void);
char data_rx;

int posisi45;
int posisi90;
int posisi110;
int posisi135;

char naik(char servo_no)
{
posisi45 = posisi45 | servo_no;
posisi90 = posisi90 | servo_no;
posisi110 = posisi110 | servo_no;
posisi135 = posisi135 | servo_no;
}

char turun(char servo_no)
{
posisi135 = posisi135 & (~servo_no);
posisi110 = posisi110 & (~servo_no);
posisi90 = posisi90 & (~servo_no);
posisi45 = posisi45 & (~servo_no);
```

```
}


char tengah(char servo_no)
{
posisi135 = posisi135 & (~servo_no);
posisi110 = posisi110 & (~servo_no);
posisi45 = posisi45 | servo_no;
posisi90 = posisi90 | servo_no;
}

#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;
      };
   };
```

```
  data_rx = data;
}

#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


// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here
PORTB = 0xFF;
delay_us(1050);
PORTB = posisi45;
delay_us(250);
PORTB = posisi90;
delay_us(200);
PORTB = posisi110;
delay_us(200);
PORTB = posisi135;
delay_us(250);
PORTB = 0x00;
}

// Declare your global variables here

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

// Crystal Oscillator division factor: 1
#pragma optsize-
CLKPR=0x80;
CLKPR=0x00;
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

// Input/Output Ports initialization
// Port A initialization
```

```
// Func2=In Func1=In Func0=In
// State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

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

// Port D initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// 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
// OC0A output: Disconnected
// OC0B output: Disconnected
TCCR0A=0x00;
TCCR0B=0x00;
TCNT0=0x00;
OCR0A=0x00;
OCR0B=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: CTC top=OCR1A
// 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: On
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0B;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0xD8;
OCR1AH=0x09;
OCR1AL=0xC4;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
```

```
// Interrupt on any change on pins PCINT0-7: Off
GIMSK=0x00;
MCUCR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x40;

// Universal Serial Interface initialization
// Mode: Disabled
// Clock source: Register & Counter=no clk.
// USI Counter Overflow Interrupt: Off
USICR=0x00;

// 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=0x06;
UBRRH=0x00;
UBRRL=0x33;

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

// Watchdog Timer initialization
// Watchdog Timer Prescaler: OSC/1024k
// Watchdog Timer interrupt: Off
#pragma optsize-

WDTCR=0x39;
WDTCR=0x29;
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

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



while (1)
     {
switch (data_rx)
{
       case 'A':
       posisi_1();
   break;
       case 'D':
       posisi_4();
```

```
   break;
      case 'T':
      diem();
   break;
      case 'L':
      muter_L();
   break;
      case 'R':
      muter_R();
   break;


   default:
   tengah(0xFF);
};
     // Place your code here
//servo 1 0x80
//servo 2 0x40
//servo 3 0x20
//servo 4 0x10
//servo 5 0x08
//servo 6 0x04
//servo 7 0x02
//servo 8 0x01


     };


//DIAM//
}
char diem(void)
  {

tengah(0XFF);

   }


//MAJU//
char posisi_1(void)
  {

tengah(0XFF);

turun(0x08);
delay_ms(0);
naik(0x02);
delay_ms(200);
naik(0x80);
delay_ms(0);
naik(0x20);
delay_ms(200);
naik(0x08);
delay_ms(0);
```

```
turun(0x02);
delay_ms(200);

tengah(0XFF);

naik(0x04);
delay_ms(0);
turun(0x01);
delay_ms(200);
turun(0x40);
delay_ms(0);
turun(0x10);
delay_ms(200);
turun(0x04);
delay_ms(0);
naik(0x01);
delay_ms(200);
  }


//MUNDUR//
char posisi_4(void)
{
tengah(0XFF);

naik(0x04);
delay_ms(0);
turun(0x01);
delay_ms(200);
turun(0x40);
delay_ms(0);
turun(0x10);
delay_ms(200);
turun(0x04);
delay_ms(0);
naik(0x01);
delay_ms(200);

tengah(0XFF);

naik(0x08);
delay_ms(0);
turun(0x02);
delay_ms(200);
naik(0x80);
delay_ms(0);
naik(0x20);
delay_ms(200);
turun(0x08);
delay_ms(0);
naik(0x02);
delay_ms(200);
}
```

```
//HADAP KANAN//
char muter_R(void)
{
tengah(0xFF);
turun(0x80);
delay_ms(150);
turun(0x08);
delay_ms(150);
naik(0x80);
delay_ms(150);
tengah(0x80);
delay_ms(150);

naik(0x40);
delay_ms(150);
turun(0x04);
delay_ms(150);
turun(0x40);
delay_ms(150);
tengah(0x40);
delay_ms(150);

turun(0x20);
delay_ms(150);
turun(0x02);
delay_ms(150);
naik(0x20);
delay_ms(150);
tengah(0x20);
delay_ms(150);

naik(0x10);
delay_ms(150);
turun(0x01);
delay_ms(150);
turun(0x10);
delay_ms(150);
tengah(0x10);
delay_ms(150);
}


//HADAP KIRI//
char muter_L(void)
{
tengah(0xFF);
turun(0x80);
delay_ms(150);
naik(0x08);
delay_ms(150);
naik(0x80);
delay_ms(150);
tengah(0x80);
delay_ms(150);
```

```
naik(0x40);
delay_ms(150);
naik(0x04);
delay_ms(150);
turun(0x40);
delay_ms(150);
tengah(0x40);
delay_ms(150);

turun(0x20);
delay_ms(150);
naik(0x02);
delay_ms(150);
naik(0x20);
delay_ms(150);
tengah(0x20);
delay_ms(150);

naik(0x10);
delay_ms(150);
naik(0x01);
delay_ms(150);
turun(0x10);
delay_ms(150);
tengah(0x10);
delay_ms(150);
}
```

SUBPROGRAM PENGGUNAAN SENSOR ULTRASONIK (PING)

```
//program ping

//PING DEPAN//
void satu(void)
{
mulai:
    t=1;
    DDRB.1=1;
    PORTB.1=1;
    delay_us(5);
    PORTB.1=0;
    DDRB.1=0;
    PORTB.1=1 ;

    for(i=0;i<1050;i++)
     {
     if(PINB.1==1)
     goto coun;
     }
goto mulai;
coun:
      if(PINB.1==0)
      goto hitung;
      t=t+1 ;
      delay_us(1);
goto coun;
hitung:
      a=460*0.0001*t;

return;
}

//PING KANAN//
void tiga(void)
{
```

```
mulai:
     t=1;
     DDRB.3=1;
     PORTB.3=1;
     delay_us(5);
     PORTB.3=0;
     DDRB.3=0;
     PORTB.3=1 ;

     for(i=0;i<1050;i++)
       {
       if(PINB.3==1)
       goto coun;
       }
goto mulai;
coun:
       if(PINB.3==0)
       goto hitung;
       t=t+1 ;
       delay_us(1);
goto coun;
hitung:
       c=460*0.0001*t;

return;
}

//PING KIRI//
void empat(void)
{
mulai:
     t=1;
     DDRB.4=1;
     PORTB.4=1;
     delay_us(5);
     PORTB.4=0;
     DDRB.4=0;
```

```
    PORTB.4=1 ;

    for(i=0;i<1050;i++)
     {
     if(PINB.4==1)
     goto coun;
     }
goto mulai;
coun:
     if(PINB.4==0)
     goto hitung;
     t=t+1 ;
     delay_us(1);
goto coun;
hitung:
     d=460*0.0001*t;
 return;
}
```

SUBPROGRAM PENGGUNAAN SENSOR *Thermal Infra Red* (Devantech Thermal Array TPA81)

```
void thermal(void)
     {
    i2c_start();
    i2c_write(0xD0);
    i2c_write(1);

    i2c_start();
    i2c_write(0xD1);
    m=i2c_read(1);
    n=i2c_read(1);
    o=i2c_read(1);
    p=i2c_read(1);
    q=i2c_read(1);
    r=i2c_read(1);
    s=i2c_read(1);
    u=i2c_read(1);
    v=i2c_read(0);
    i2c_stop();
    delay_ms(200);
    lcd_clear();
    sprintf(text,"%2u %2u %2u %2u %2u  %2u %2u %2u %2u",m,n,o,p,q,r,s,u,v);
    lcd_puts(text);
    delay_ms(500);
    }
```

LAMPIRAN C
DATASHEET

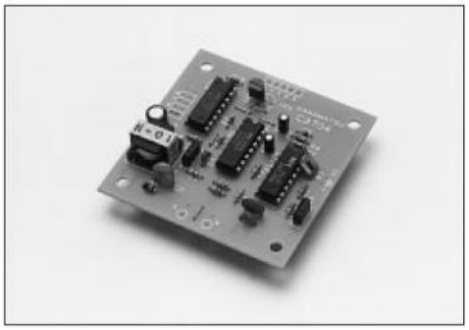
SENSOR API LILIN (HAMAMATSU UVTRON)

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**

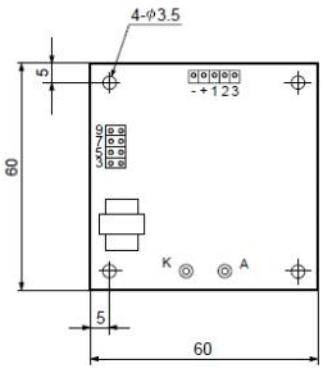
SPECIFICATIONS

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)

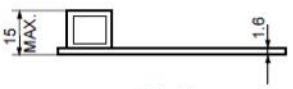
	C3704	C3704-02	C3704-03
Input Voltage	10 to 30 Vdc	5Vdc ± 5%	6 to 9 Vdc
Current consumption	3 mA Max.	300µA Max.	300µA Max.

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>

TPT A0024EA

Figure 2: Schematic Diagram

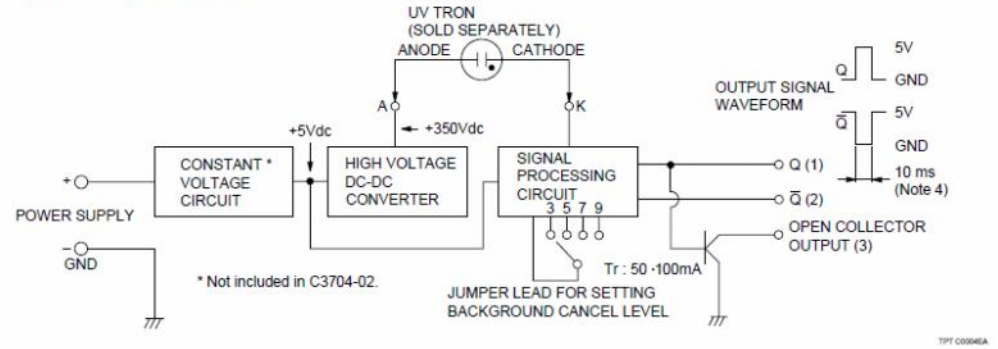
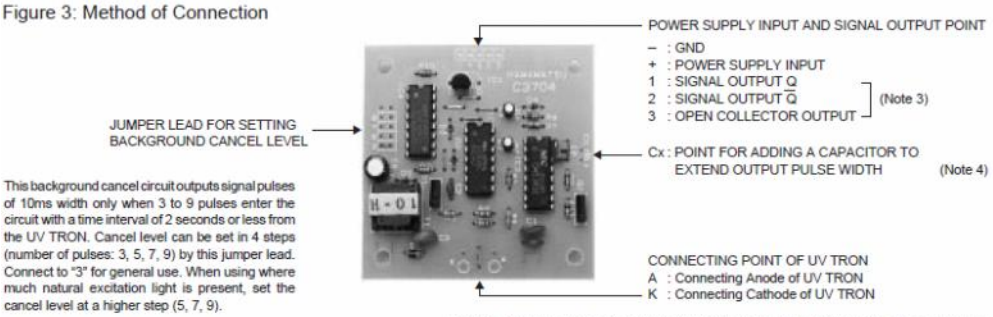


Figure 3: Method of Connection



This background cancel circuit outputs signal pulses of 10ms width only when 3 to 9 pulses enter the circuit with a time interval of 2 seconds or less from the UV TRON. Cancel level can be set in 4 steps (number of pulses: 3, 5, 7, 9) by this jumper lead. Connect to "3" for general use. When using where much natural excitation light is present, set the cancel level at a higher step (5, 7, 9).

Note 3: No load can be driven by an output from points "1" and "2" because these signals are output from the only C-MOS IC directly. When a load such as a buzzer and a relay is connected to this circuit, it should be connected to the point open collector output. The transistor ratings of the open collector is 50V, 100mA. Be careful not to exceed the ratings.

Note 4: The output pulse width is set to 10ms at shipping. If the pulse width needs to be extended, add a capacitor to this point. (When using an electrolytic condenser, make sure the polarity is correct.)
e.g. CX = 1 µF: Pulse Width ≅ 1s, CX=10 µF : Pulse Width ≅ 10s

PRECAUTIONS FOR USE

- Since the operation impedance is extremely high, the UV TRON should be connected as close as possible to the circuit board within 5 cm.
- Take care to avoid external noise since a C-MOS IC is used in the circuit. It is recommended that the whole PC board be put in the shield box when it is used.
- To reduce current consumption, oscillating frequency is very low (approx. 20 Hz) in this DC-DC converter. Thus, the output impedance of the high voltage power supply is extremely high. If the surrounding humidity is high, electrical leakage on the PC board surface may lead to a drop in the supply voltage to the UV TRON. This voltage drop may result in lowered detection performance, so a moistureproof material (silicone compound, etc.) should be applied at the connecting point of the UV TRON, etc., if using the unit in a humid environment.

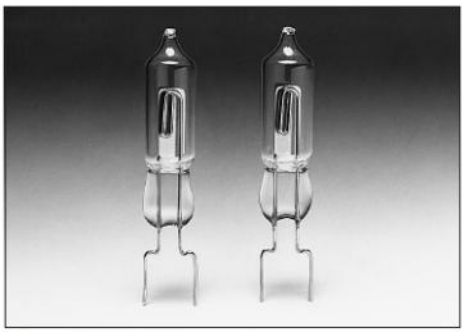
- A model equipped with a flame sensor (R2868) is also available.

Quick Detection of Flame from Distance, Compact UV Sensor with High Sensitivity and Wide Directivity, Suitable for Flame Detectors and Fire Alarms.

Hamamatsu R2868 is a UV TRON ultraviolet detector that makes use of the photoelectric effect of metal and the gas multiplication effect. It has a narrow spectral sensitivity of 185 to 260 nm, being completely insensitive to visible light. Unlike semiconductor detectors, it does not require optical visible-cut filters, thus making it easy to use.

In spite of its small size, the R2868 has wide angular sensitivity (directivity) and can reliably and quickly detect weak ultraviolet radiations emitted from flame due to use of the metal plate cathode (eg. it can detect the flame of a cigarette lighter at a distance of more than 5 m.).

The R2868 is well suited for use in flame detectors and fire alarms, and also in detection of invisible discharge phenomena such as corona discharge of high-voltage transmission lines.



APPLICATIONS

- **Flame detectors for gas/oil lighters and matches**
- **Fire alarms**
- **Combustion monitors for burners**
- **Inspection of ultraviolet leakage**
- **Detection of discharge**
- **Ultraviolet switching**

GENERAL

Parameters	Rating	Units
Spectral Response	185 to 260	nm
Window Material	UV glass	—
Weight	Approx. 1.5	g
Dimensional Outline	See Fig. 3	—

MAXIMUM RATINGS

Parameters	Rating	Units
Supply Voltage	400	Vdc
Peak Current [1]	30	mA
Average Discharge Current [2]	1	mA
Operating Temperature	-20 to +60	°C

CHARACTERISTICS (at 25°C)

Parameters	Rating	Units
Discharge Starting Voltage (with UV radiation)	280	Vdc Max.
Recommended Operating Voltage	325±25	Vdc
Recommended Average Discharge Current	100	µA
Background [3]	10	cpm Max
Sensitivity [4]	5000	cpm Typ.

NOTES:

1) This is the maximum momentary current that can be handled if its full width at half maximum is less than 10 µs.
2) If the tube is operated near this or higher, the service life is noticeably reduced. Use the tube within the recommended current values.
3) Measured under room illuminations (approximately 500 lux) and recommended operating conditions. Note that these values may increase if the following environmental factors are present.
   1. Mercury lamps, sterilization lamps, or halogen lamps are located nearby.
   2. Direct or reflected sunlight is incident on the tube.
   3. Electrical sparks such as welding sparks are present.
   4. Radiation sources are present.
   5. High electric field (including static field) generates across the tube.
4) These are representative values for a wavelength of 200 nm and a light input of 10 pW/cm². In actual use, the sensitivity will vary with the wavelength of the ultraviolet radiation and the drive circuitry employed.

Figure 1: UV TRON's Spectral Response and Various Light Sources

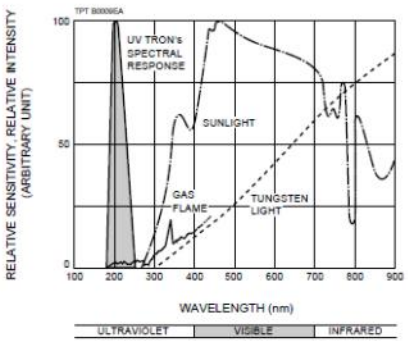


Figure 2: Angular Sensitivity (Directivity)

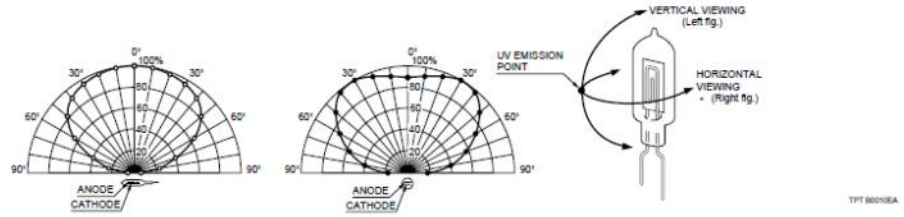


Figure 3: Dimensional Outline (Unit: mm)

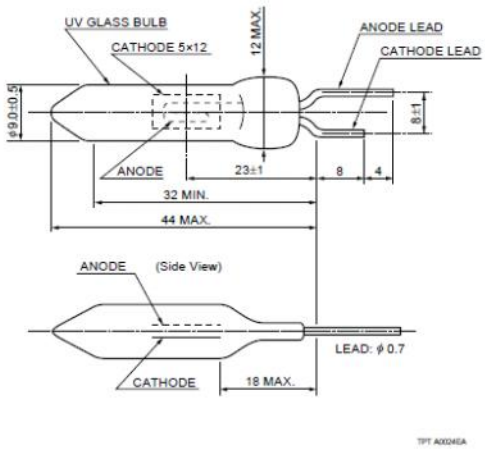
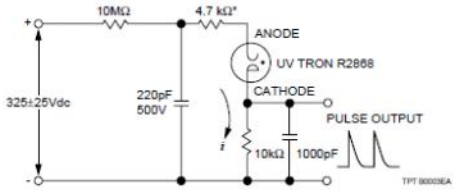
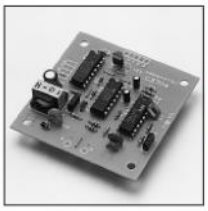


Figure 4: Recommended Operating Circuit



* 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.

PRECAUTIONS FOR USE

- **Ultraviolet Radiation**

The UV TRON itself emits ultraviolet radiation in operation. When using two or more UV TRONs at the same time in close position, care should be taken so that they do not optically interfere with each other.

- **Vibration and Shock**

The UV TRON is designed in accordance with the standards of MIL-STD-202F (Method 204D/0.06 inch or 10g, 10- 500Hz, 15 minutes, 1 cycle) and MIL-STD-202F (Method 213B/100g, 11ms, Half-sine, 3 times). However, should a strong shock be sustained by the UV TRON (e.g. if dropped), the glass bulb may crack or the internal electrode may be deformed, resulting in deterioration of electrical characteristics. So extreme care should be taken in handling the tube.

- **Polarity**

Connect the UV TRON with correct polarity. Should it be connected with reverse polarity, operating errors may occur.

WARRANTY.

The UV TRON is covered by a warranty for a period of one year after delivery. The warranty is limited to replacement of any defective tube due to defects traceable to the manufacturer.

SENSOR JARAK ULTRASONIK (PING)

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

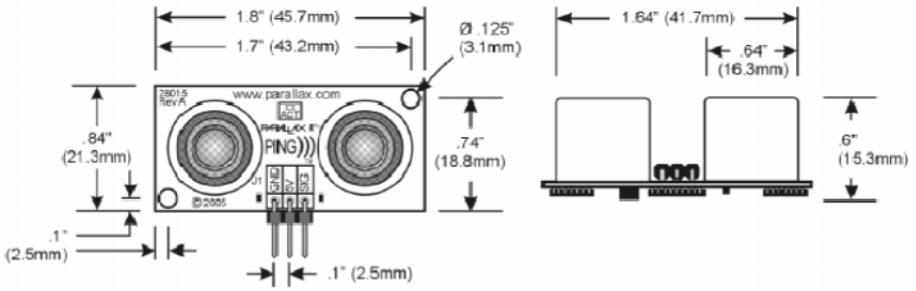
The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamp® or Javelin Stamp modules, SX or Propeller chips, or other microcontrollers, requiring only one I/O pin.

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

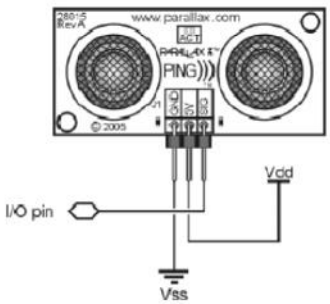
- Supply voltage – +5 VDC
- Supply current – 30 mA typ; 35 mA max
- Range – 2 cm to 3 m (0.8 inches to 3.3 yards)
- Input trigger – positive TTL pulse, 2 µs min, 5 µs typ.
- Echo pulse – positive TTL pulse, 115 µs to 18.5 ms
- Echo hold-off – 750 µs from fall of trigger pulse
- Burst frequency – 40 kHz 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 – 70º C.

Dimensions



Pin Definitions

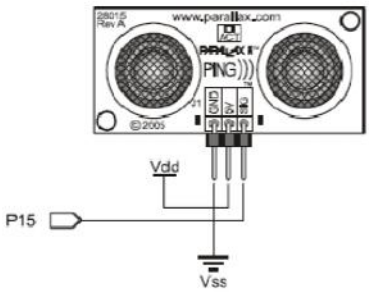
GND	Ground (Vss)
5 V	5 VDC (Vdd)
SIG	Signal (I/O pin)

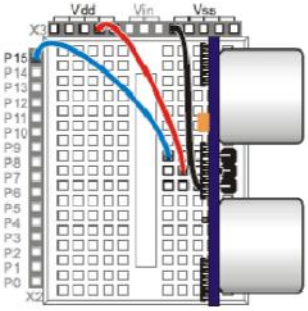


The PING))) sensor has a male 3-pin header used to supply ground, power (5 VDC) and signal. The header allows the sensor to be plugged into a solderless breadboard, or to be located remotely through the use of a standard servo extender cable (Parallax part #805-00002). Standard connections are shown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

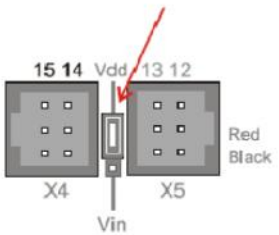




Extension Cable and Port Cautions

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

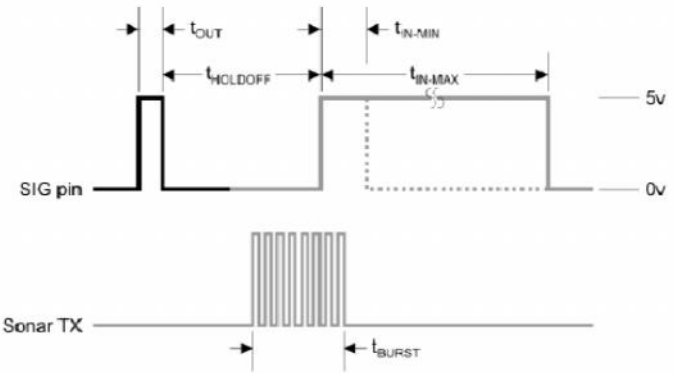
1. When plugging the cable onto the PING))) sensor, connect Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown. Then plug the cable into the 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.



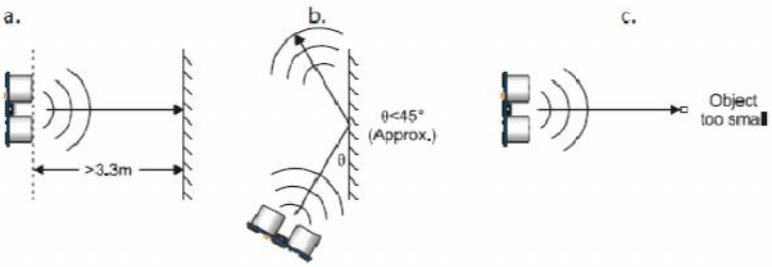
Host Device
PING))) Sensor

t_{OUT}	2 µs (min), 5 µs typical
$t_{HOLDOFF}$	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.



Target Object Material

In addition, objects that absorb sound or have a soft or irregular surface, such as a stuffed animal, may not reflect enough sound to be detected accurately. The PING))) sensor will detect the surface of water, however it is not rated for outdoor use or continual use in a wet environment. Condensation on its transducers may affect performance and lifespan of the device. See the Water Level with PING))) document on the 28015 product page.

Air Temperature

Temperature has an effect on the speed of sound in air that is measurable by the PING))) sensor. If the temperature (°C) is known, the formula is:

$$C_{air} = 331.5 + (0.6 \times T_C)\ m/s$$

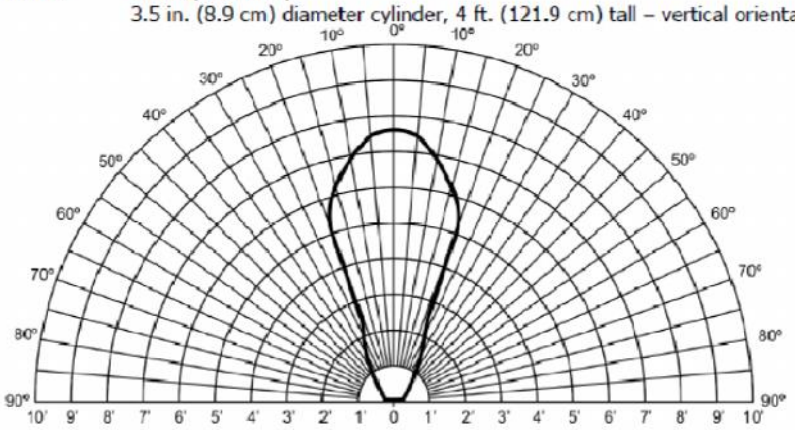
The percent error over the sensor's operating range of 0 to 70 ° C is significant, in the magnitude of 11 to 12 percent. The use of conversion constants to account for air temperature may be incorporated into your program (as is the case in the BS2 program below). Percent error and conversion constant calculations are introduced in Chapter 2 of *Smart Sensors and Applications*, a Stamps in Class text available for download from the 28029 product page.

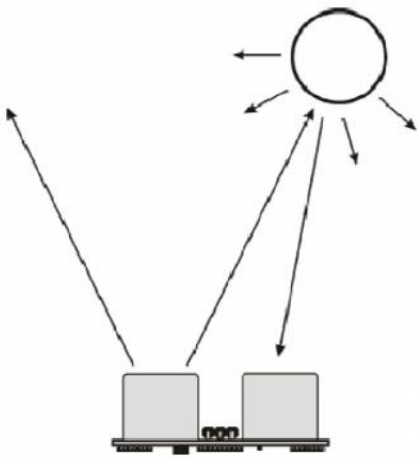
Test Data

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

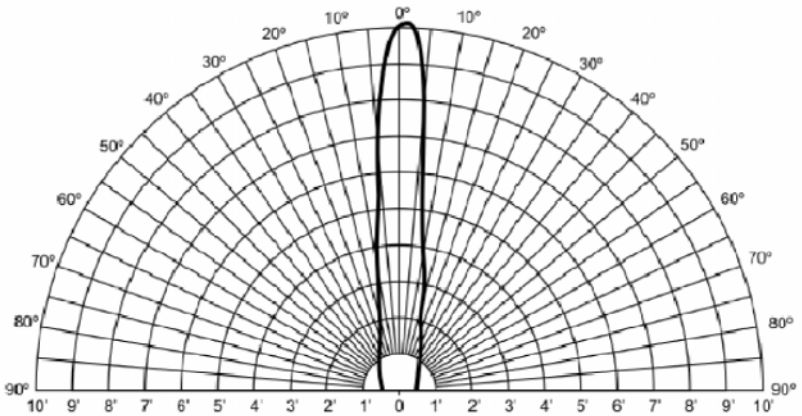
Sensor Elevation: 40 in. (101.6 cm)
Target: 3.5 in. (8.9 cm) diameter cylinder, 4 ft. (121.9 cm) tall – vertical orientation

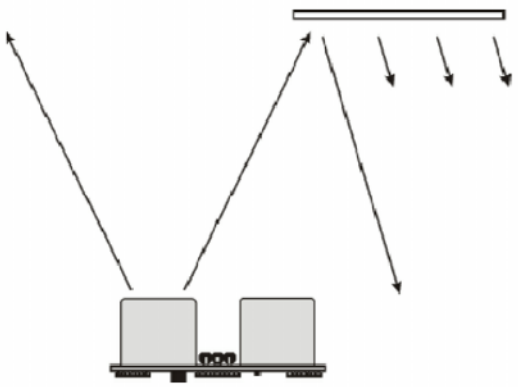




Test 2

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





SENSOR THERMAL INFRARED

(THERMAL ARRAY TPA 81)

TPA81 Thermopile Array

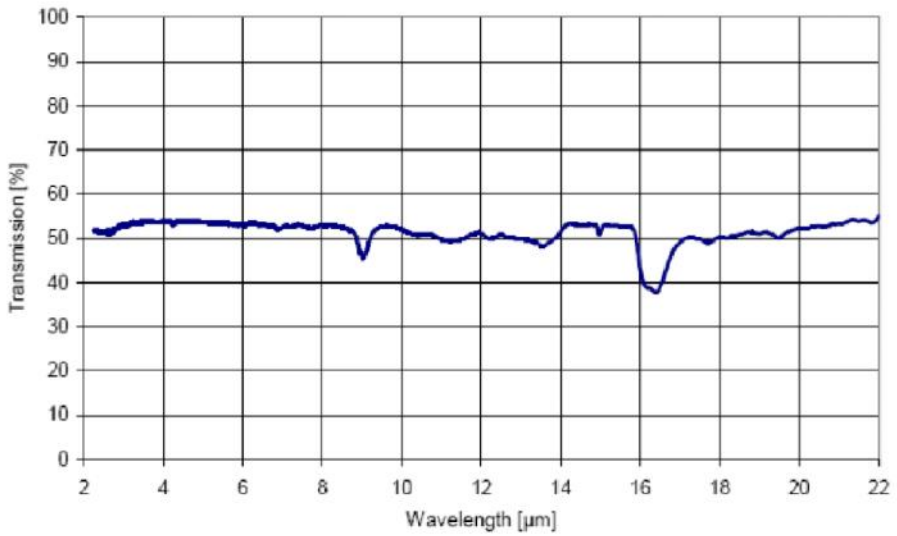
Technical Specification

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 (FOV)

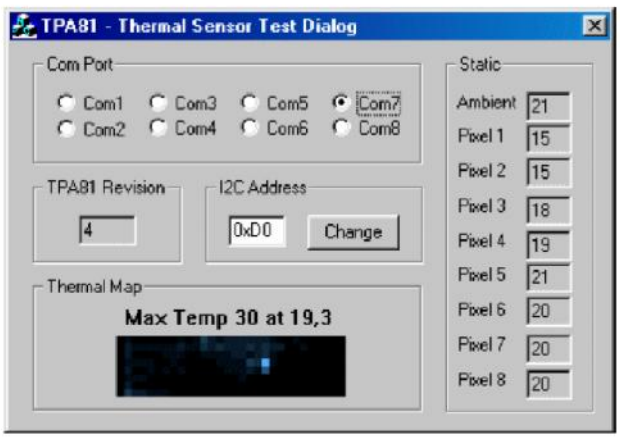
The typical field of view of the TPA81 is 41° by 6° making each of the eight pixels 5.12° by 6°. The array of eight pixels is orientated along the length of the PCB - that's from top to bottom in the diagram below. Pixel number one is nearest the tab on the sensor - or at the bottom in the diagram below.

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.

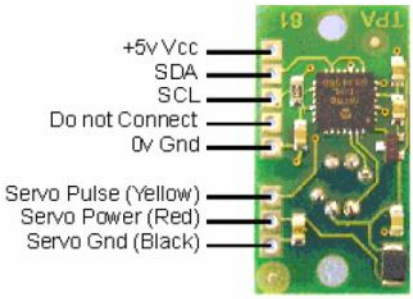
The following is a snapshot of our test program. It displays a 32x8 bitmap produced by using a servo to pan the sensor. If you want a copy of this windows based program, its here, but you will need an RF04/CM02 to connect the TPA81 to your PC. Here you can see a candle flame about a meter away showing 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 50mS 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.