

LAMPIRAN A
FOTO BALON TERBANG

Tampak Bawah



Tampak Depan



LAMPIRAN B
PROGRAM PADA PENGONTROL MIKRO

```
/******
```

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

Project :
Version :
Date : 11/03/2011
Author : F4CG
Company : F4CG
Comments:

Chip type : ATmega16
Program type : Application
Clock frequency : 11,095200 MHz
Memory model : Small
External SRAM size : 0
Data Stack size : 256

```
*****/
```

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

```
unsigned int a,b,c,x,s,t,i,e,f,g,h;  
unsigned char dat[20],text[32];  
eeprom unsigned int d;  
//eeprom unsigned char  
dat2[50],dat3[50],dat4[50],dat5[50],dat6[50],dat7[50],dat8[50],dat9[50],tinggi[50];  
unsigned char  
dat2[50],dat3[50],dat4[50],dat5[50],dat6[50],dat7[50],dat8[50],dat9[50],tinggi[50];  
// Alphanumeric LCD Module functions
```

```
#asm  
    .equ __lcd_port=0x15 ;PORTC  
#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;
};
};
if(data==255)// carry return
{
x=0;
s=1;
}
if(s==1)
{
dat[x]=data;
x++;
}
}

```

```

        if(x>10) s=0;
    }
}

#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>

// Declare your global variables here
void ping(void)    //progrm ping
{
    mulai:
        t=1;
        DDRA.7=1;
        PORTA.7=1;
        delay_us(5);
        PORTA.7=0;
        DDRA.7=0;
        PORTA.7=1;

        for(i=0;i<1050;i++)
        {
            if(PINA.7==1)
                goto coun;
        }
    goto mulai;
coun:
    if(PINA.7==0)
        goto hitung;
}

```

```

        t=t+1 ;
        delay_us(1);
goto coun;
hitung:
        a=460*0.0001*t;

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=0x40;
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=0xFF;

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

// 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: On

```

```

// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 115200
UCSRA=0x00;
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x05;

// 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")
d=0;
printf("RS\r"); // Reset the camera
delay_ms(100); // must delay ...can delay less than 100ms
printf("\r");
delay_ms(100);
printf("cr 18 36\r");
delay_ms(100);
printf("PM 0\r"); // turn poll mode on
delay_ms(100);
printf("RM 1\r"); // turn on raw data mode for packets received from camera
delay_ms(100);
printf("SW 1 1 80 143\r"); // not necessary this is default window
delay_ms(100);
printf("tc 20 240 20 100 20 100\r");//@#$%^&*percobaab
delay_ms(100);

while (1)
{
// Place your code here
if(PINA.6==0)
{
PORTB.7=0;//enable 21
PORTD.4=0;//enable 1a
PORTD.7=0;//enable 1b
lcd_clear();
}
}

```

```

    printf("%d %d %d %d %d %d %d %d %d
%d", dat2[d], dat3[d], dat4[d], dat5[d], dat6[d], dat7[d], dat8[d], dat9[d], tinggi[d]);
    sprintf(text, "%d %d %d %d %d %d %d %d
%d", dat2[d], dat3[d], dat4[d], dat5[d], dat6[d], dat7[d], dat8[d], dat9[d], tinggi[d]);
    lcd_puts(text);
    delay_ms(500);
    d++;
    if (d>=49)
    {
        lcd_clear();
        printf("SELESAI");
        sprintf(text, "SELESAI");
        lcd_puts(text);
        delay_ms(10000);
        d=0;
    }
}
else
{

```

```

ping();
lcd_clear();
sprintf(text, "%d %d %d %d %d %d %d %d %d %d
%d", dat[2], dat[3], dat[4], dat[5], dat[6], dat[7], dat[8], dat[9], a, g, d);
lcd_puts(text);
//program simpan data
g++;
if(g>=2)
{
    g=0;
    dat2[d]=dat[2];
    dat3[d]=dat[3];
    dat4[d]=dat[4];
    dat5[d]=dat[5];
    dat6[d]=dat[6];
    dat7[d]=dat[7];
    dat8[d]=dat[8];
    dat9[d]=dat[9];
    tinggi[d]=a;
    d++;
    if(d>=49)
    {
        PORTB.7=0;//enable 21
        PORTD.4=0;//enable 1a
        PORTD.7=0;//enable 1b
    }
}

```

```

    lcd_clear();
    sprintf(text,"selesai");
    lcd_puts(text);
    delay_ms(100000);
}
}
if(dat[3]>=46 && dat[3]<96)
{
    if (a<150) //naik
    {
        lcd_clear();
        sprintf(text,"naik");
        lcd_puts(text);
        for(e=0;e<100;e++)
        {
            PORTB.5=1;
            PORTB.6=0;
            PORTB.7=1;//enable 21
            delay_ms(1);
            PORTB.7=0;//enable 2a
            delay_ms(10);
        }
    }
    else //turun
    {
        lcd_clear();
        sprintf(text,"turun");
        lcd_puts(text);
        for(f=0;f<100;f++)
        {
            PORTB.5=0;
            PORTB.6=1;
            PORTB.7=1;//enable 21
            delay_ms(1);
            PORTB.7=0;//enable 2a
            delay_ms(10);
        }
    }
}
else if(dat[3]>=96) //maju
{
    lcd_clear();
    sprintf(text,"%d %d maju",dat[2],dat[3]);
    lcd_puts(text);
    for(c=0;c<100;c++)
    {

```

```

PORTD.2=1;
PORTD.3=0;
PORTD.4=1;//enable 1a
PORTD.5=1;
PORTD.6=0;
PORTD.7=1;//enable 1b
delay_ms(1);
PORTD.4=0;//enable 1a
PORTD.7=0;//enable 1a
delay_ms(10);
}
}
else if(dat[3]<46) //mundur
{
lcd_clear();
sprintf(text,"%d %d mundur",dat[2],dat[3]);
lcd_puts(text);
for(b=0;b<100;b++)
{
PORTD.2=0;
PORTD.3=1;
PORTD.4=1;//enable 1a
PORTD.5=0;
PORTD.6=1;
PORTD.7=1;//enable 1b
delay_ms(1);
PORTD.4=0;//enable 1a
PORTD.7=0;//enable 1a
delay_ms(10);
}
}

//koreksi x
if(dat[2]>=69) //kanan
{
lcd_clear();
sprintf(text,"%d %d kanan",dat[2],dat[3]);
lcd_puts(text);
for(b=0;b<300;b++)
{
PORTD.2=1;
PORTD.3=0;
PORTD.4=1;//enable 1a
PORTD.5=0;
PORTD.6=1;

```

```

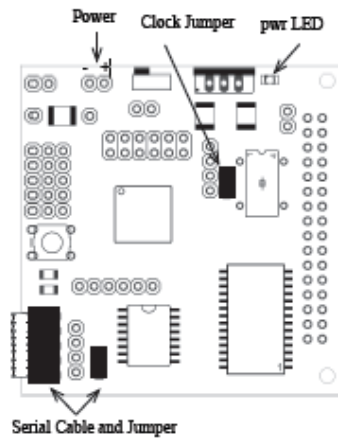
    PORTD.7=1;//enable 1b
    delay_ms(1);
    PORTD.4=0;//enable 1a
    PORTD.7=0;//enable 1a
    delay_ms(10);
}
}
if(dat[2]<18) //kiri
{
    lcd_clear();
    sprintf(text,"%d %d kiri",dat[2],dat[3]);
    lcd_puts(text);
    for(c=0;c<300;c++)
    {
        PORTD.2=0;
        PORTD.3=1;
        PORTD.4=1;//enable 1a
        PORTD.5=1;
        PORTD.6=0;
        PORTD.7=1;//enable 1b
        delay_ms(1);
        PORTD.4=0;//enable 1a
        PORTD.7=0;//enable 1a
        delay_ms(10);
    }
}
}
};
}

```

LAMPIRAN C
DATASHEET

Setting Up the Hardware

In order to initially test your CMUcam2, you will need a serial cable, a power adapter and a computer. The CMUcam2 can use a power supply which produces anywhere from 6 to 15 volts of DC power capable of supplying at least 200mA of current. This can be provided by either an AC adapter (possibly included) or a battery supply. These should be available at any local electronics store. The serial cable should have been provided with your CMUcam2. Make sure that you have the CMOS sensor board connected to the CMUcam2 board so that it is in the same orientation as the picture shows on the cover of this manual.



First, connect the power. Make sure that the positive side of your power plug is facing away from the main components on the board. If the camera came with an AC adapter, make sure that the connector locks into the socket correctly.

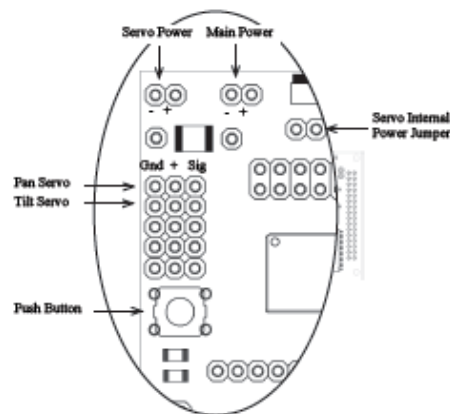
Now that the camera has power, connect the serial link between the camera and your computer. This link is required initially so that you can test and focus your camera. The serial cable should be connected so that the ribbon part of the cable faces away from the board. You must also connect the serial pass through jumper.

Check to make sure that the clock jumper is connected. This allows the clock to actively drive the processor.

Once everything is wired up, try turning the board on. The power LED should illuminate green and only one LED should remain on. Both LEDs turn on upon startup, and one turns off after the camera has been successfully configured.

Demo Mode

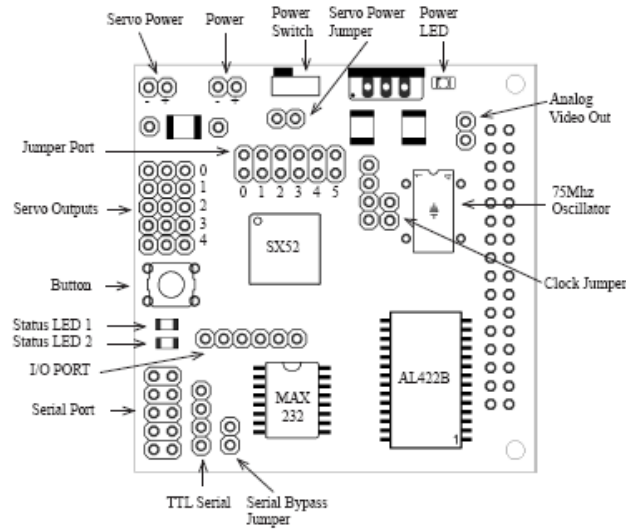
Demo mode causes the camera to call track window and then drive two standard hobby servos towards the object being tracked. This can be initiated autonomously at startup. First you need to plug a pan and/or tilt servo into servo ports 0 and 1. Servo port 0 is for the pan, while 1 is for the tilt. Next, make sure that the servos are being powered by either the internal servo power jumper or by an external power source. While holding down the push button, turn the camera on. The tracking LED should begin rapidly blinking. Immediately release the push button and wait for the LED to stop blinking. Next, point the camera at a colored object and press the push button again. This should grab the color of the object and begin automatically servoing towards it. If the servos appear to be driving in the reverse direction, add the appropriate servo direction jumper. During the period when the LED is blinking, the camera is adjusting to the light conditions in the room. Try not to hold the object in front of the camera while this is occurring. Experiment with different colors and lighting. You will notice that some work much better than others.



The following steps are performed during power up in demo mode:

1. RS is sent to the camera
2. Pause 5 seconds while blinking the LED to allow the camera to stabilize
3. The Camera register string "CR 18 32 19 32" is sent.
4. Auto Servo Mode is enabled.
5. TW is called.

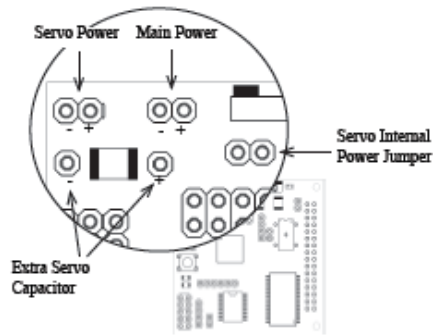
Board Layout



Ports

Power

The input power to the board goes through a 5 volt regulator. It is ideal to supply the board with between 6 and 15 volts of DC power that is capable of supplying at least 200 milliamperes of current.



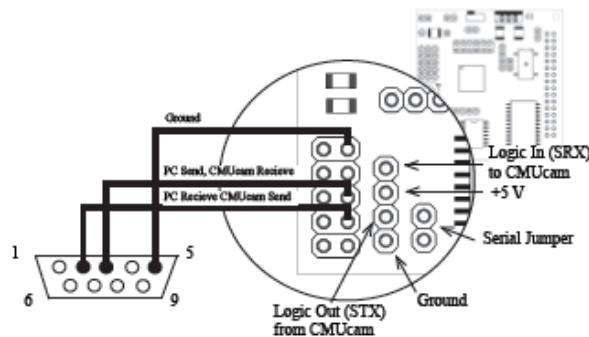
The servos can either be powered by internal power, or by the external servo power connector. To run them off of internal power, connect a jumper across the "servo internal power jumper". To run them off of external power, leave the jumper open, and connect another 5volt supply to the servo power connector. Do not connect an external servo supply while the servo power jumper is in place. If the servos are drawing too much power or seem to be noisy, try soldering a large valued capacitor across the "Extra Servo Capacitor" connections. The external servo power is not switched by the main power switch.

Serial Port

The CMUcam2 has a standard level shifted serial port to talk to a computer as well as a TTL serial port for talking to a microcontroller.

The level shifted serial port only uses 3 of the 10 pins. It is in a 2x5 pin configuration that fits a standard 9 pin ribbon cable clip-on serial sockets and 10 pin female clip on serial headers that can both attach to a 10 wire ribbon cable. If this initially does not work, try flipping the direction that the ribbon cable connects to the CMUcam2 board. Make sure the serial jumper is in place when you use this mode.

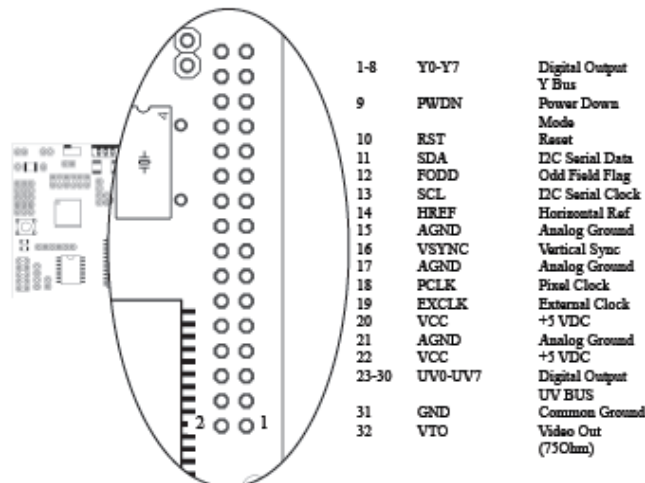
The TTL connector can be used to talk to a microcontroller without the use of a level shifting chip. It operates between 0 and 5 volts. Remove the Serial Jumper when you use this mode.



The Trepanhead serial connector shown is what the serial connector on your computer should look like if drawn in an annoying line art drawing program.

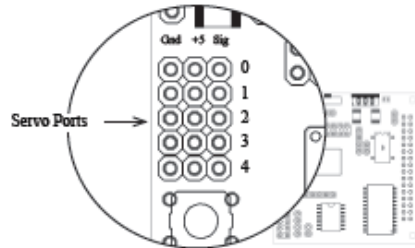
Camera Bus

This bus interfaces with the CMOS camera chip. The CMOS camera board is mounted parallel to the processing part of the board and connects starting at pin 1. The female camera header should be soldered on the back of the board.



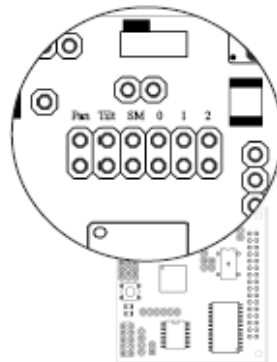
Servo Port

The CMUcam2 has the ability to control 5 servos. This can be useful if you do not wish to use a separate servo controller. The servo port can also be used as a general purpose digital outputs.



Configuration Jumpers

The jumpers can be used to set the camera's baudrates or configure various modes of operation.



Jumpers 0 1 and 2 set the camera into the following baudrates:

Baud Rate	Pin 0 1 2
115,200 Baud	--_
57,600 Baud	--X
38,400 Baud	-X_
19,200 Baud	-X X
9,600 Baud	X__
4,800 Baud	X_X
2,400 Baud	XX_
1,200 Baud	XXX

X - jumper closed
_ - jumper open

Pan and Tilt Reverse Jumpers

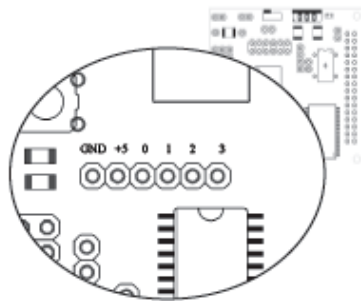
During Auto Servo Mode, or demo mode it may be necessary to reverse the direction of the pan or tilt servo. Connecting the pan and/or tilt jumper will cause auto servo mode to send the opposite commands to each servo. Note, this only works for auto servo mode, and not for normal servo operations.

Slave Mode Jumper

The CMUcam2 supports a mode of operation that allows multiple boards to process data from the same camera. If a PC104 style pass-through header is used instead of the standard double row female header, it is possible to rack multiple boards along the same camera bus. Upon startup, if the "SM" jumper is set, the camera becomes a slave. Slave mode stops the camera board from being able to configure or interfere with the CMOS camera's settings. Instead it processes the format setup by the master vision board. When linking the buses together you must only have one master; all other boards should be setup to be in slave mode. In this current version of the system there is no message passing between boards other than the image data from the camera bus. This means you have to communicate to each slave board via a separate serial link. This communication to the board should be identical to using a single CMUcam2. For example, you could have the master board tracking some color while the slave board could be told to get mean color data. Each board runs independently of one another and only the master can control camera registers.

Axuliary I/O

The CMUcam has 4 auxiliary Input Output ports that can be used for reading data from external devices. Note, that pin 3 is used for the sleep deeply command.

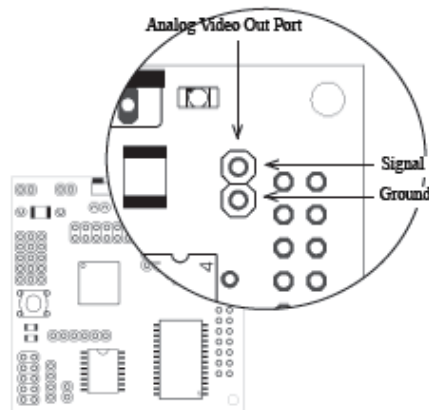


Analog Video Port

Using the OV6620 camera module, you will be able to get a PAL video signal from the analog port of the CMUcam2. This would sync up with any PAL monitor, but will not work with a standard NTSC monitor.

The OV7620 camera module will output a standard black and white NTSC video signal.

To use this output, it is necessary to keep the camera at its maximum frame rate (the default) and switch it into YCrCb mode in order to see the image on a monitor.



Functionally Grouped Command Listing

Buffer Commands

BM	Buffer Mode	30
RF	Read Frame	47

Camera Module Commands

CR	Camera Register	31
CP	Camera Power	32
CT	Camera Type	32

Data Rate Commands

DM	Delay Mode	33
PM	Poll Mode	46
PS	Packet Skip	47
RM	Raw Mode	48
PF	Packet Filter	46
OM	Output Packet Mask	45

Servo Commands

SV	Servo Position	53
SP	Servo Parameters	52
GS	Get Servo Position	36
SM	Servo Mask	51
SO	Servo Output	51

Image Windowing Commands

SF	Send Frame	50
DS	Down Sample	33
VW	Virtual Window	55
FS	Frame Stream	34
HR	HiRes Mode	38
GW	Get Window	38
FD	Pixel Difference	46

Auxiliary I/O Commands

GB	Get Button	35
GI	Get Auxiliary I/O	35
LO(1)	LED control	39

Color Tracking Commands

TC	Track Color	53
TI	Track Inverted	53
TW	Track Window	54
NF	Noise Filter	44
LM	Line Mode	40
GT	Get Tracking Parameters	37
ST	Set Tracking Parameters	52

Histogram Commands

GH	Get Histogram	35
HC	Histogram Config	38
HT	Histogram Track	39

Frame Differencing Commands

FD	Frame Difference	34
DC	Difference Channel	32
LP	Load Frame	39
MD	Mask Difference	44
UD	Upload Difference	55
HD	HiRes Difference	38
LM	Line Mode	40

Color Statistics Commands

GM	Get Mean	36
LM	Line Mode	40

System Level Commands

SD	Sleep Deeply	49
SL	Sleep	50
RS	Reset	49
GV	Get Version	37

Alphabetical Command Listing

BM	Buffer Mode	30
CR	Camera Register	31
CP	Camera Power	32
CT	Set Camera Type	32
DC	Difference Channel	32
DM	Delay Mode	33
DS	Down Sample	33
FD	Frame Difference	34
FS	Frame Stream	34
GB	Get Button	35
GH	Get Histogram	35
GI	Get Aux I/O inputs	35
GM	Get Mean	36
GS	Get Servo Positions	36
GT	Get Tracking Parameters	37
GV	Get Version	37
GW	Get Window	38
HC	Histogram Configs	38
HD	High Resolution Difference	38
HR	HiRes Mode	38
HT	Set Histogram Track	39
LO(1)	LED Control	39
LP	Load Frame to Difference	39

LM	Line Mode	40
MD	Mask Difference	44
NF	Noise Filter	44
OM	Output Packet Mask	45
PD	Pixel Difference	46
PF	Packet Filter	46
PM	Poll Mode	46
PS	Packet Skip	47
RF	Read Frame into Buffer	47
RM	Raw Mode	48
RS	Reset	49
SD	Sleep Deeply	49
SF	Send Frame	50
SL	Sleep Command	50
SM	Servo Mask	51
SO	Servo Output	51
SP	Servo Parameters	52
ST	Set Track Command	52
SV	Servo Position	53
TC	Track Color	53
TI	Track Inverted	53
TW	Track Window	54
UD	Upload Difference buffer	55
VW	Virtual Window	55

SENSOR JARAK ULTRASONIK (PING)

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

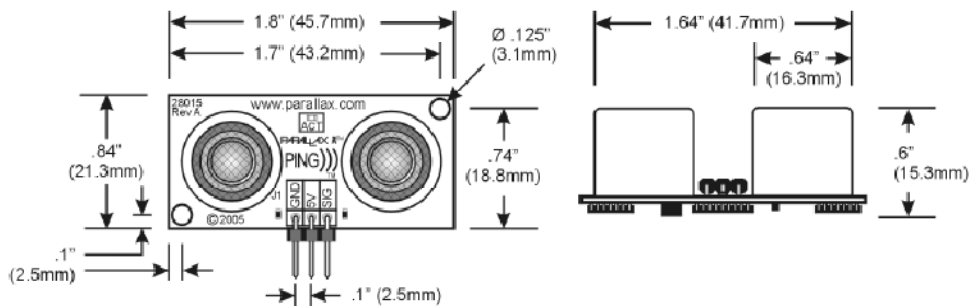
The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamp® or Javelin Stamp modules, SX or Propeller chips, or other microcontrollers, requiring only one I/O pin.

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 μ s min, 5 μ s typ.
- Echo pulse – positive TTL pulse, 115 μ s to 18.5 ms
- Echo hold-off – 750 μ s from fall of trigger pulse
- Burst frequency – 40 kHz for 200 μ s
- Burst indicator LED shows sensor activity
- Delay before next measurement – 200 μ s
- 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.

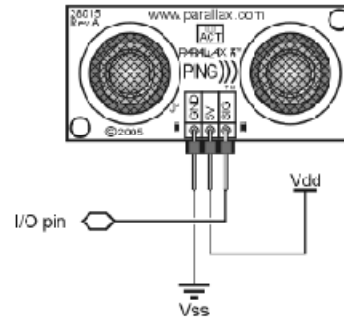
Dimensions



Pin Definitions

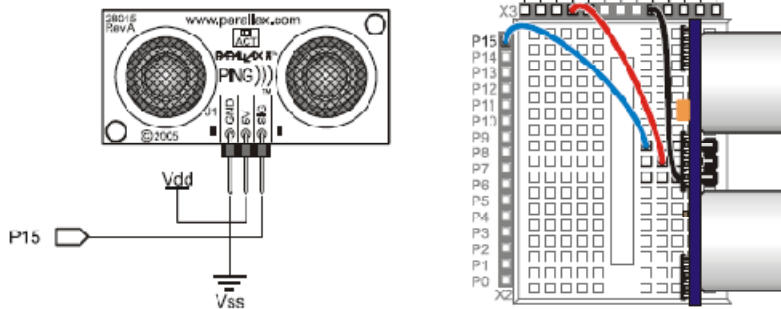
GND	Ground (Vss)
5 V	5 VDC (Vdd)
SIG	Signal (I/O pin)

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 #B05-00002). Standard connections are shown in the diagram to the right.



Quick-Start Circuit

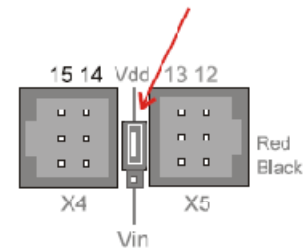
This circuit allows you to quickly connect your PING))) sensor to a BASIC Stamp[®] 2 via the Board of Education[®] breadboard area. The PING))) module's GND pin connects to Vss, the 5 V pin connects to Vdd, and the SIG pin connects to I/O pin P15. This circuit will work with the example BASIC Stamp program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

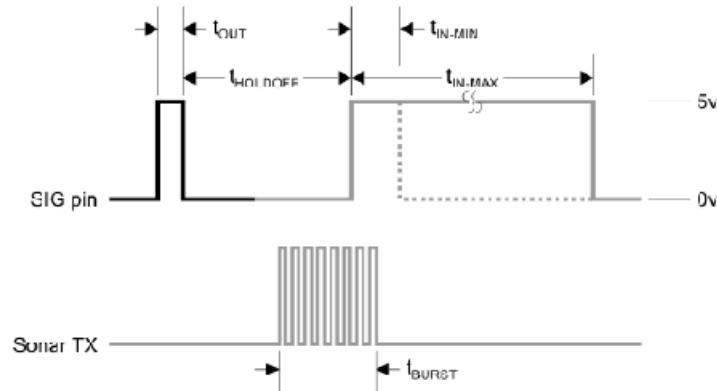
1. When plugging the cable onto the PING))) sensor, connect Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown. Then plug the cable into the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

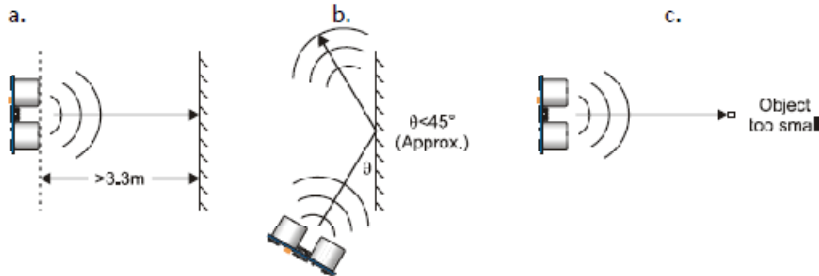


	Host Device	t_{OUT}	2 μ s (min), 5 μ s typical
	PING))) Sensor	$t_{HOLDOFF}$	750 μ s
		t_{BURST}	200 μ s @ 40 kHz
		t_{IN-MAX}	115 μ s
		t_{IN-MIN}	18.5 μ s

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 ($^{\circ}\text{C}$) is known, the formula is:

$$c_{\text{air}} = 331.5 + (0.6 \times T_c) \text{ m/s}$$

The percent error over the sensor's operating range of 0 to 70 $^{\circ}\text{C}$ 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.

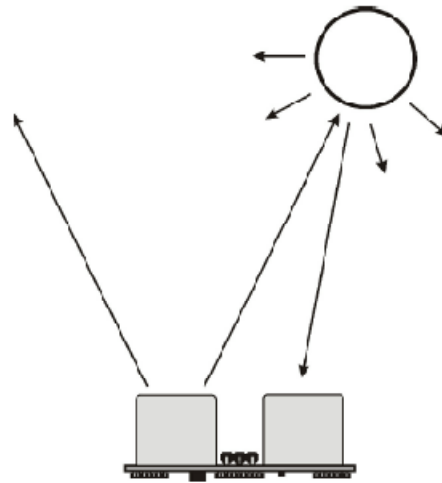
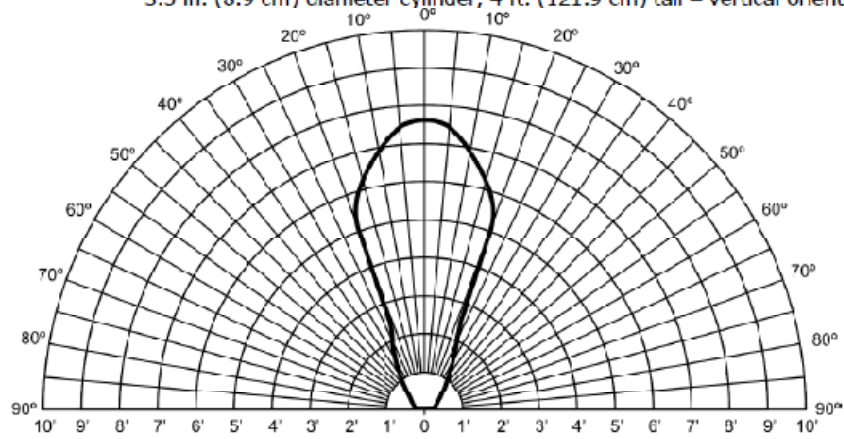
Test Data

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

