IR Timing Device

Version 1.0

Designed & Constructed by:
Peter K. Keller
psilord@cs.wisc.edu

Dedicated to the advancement of science and to my wife—who endured a lot of headaches and was of great help.

DESCRIPTION

The purpose of the IR Timing Device, henceforth "the device", is to temporally measure the occlusion/present state changes of an infrared beam to approximately 1 millisecond resolution. The device auto-calibrates to the ambient IR intensity in the area, detects the state change events, and then sends the calibration and event data to a program running on a host computer via the serial interface.

DESIGN

Power Supply

Figure 1 details the schematic for the power supply which is powered by a +9 volt 800mA power brick with a positive tip, and negative shield. Diode A1 provides reverse polarity protection and even though the voltage drop is ~0.6V, it happens before the input into the 7805 voltage regulator, thus the regulator "sees" about 8.4 V as input. A2 smoothes the incoming power to A3 and is the values are picked such that the farad value is more than twice what the possible current draw of the device may be, and the voltage rating is more than twice possible of the input voltage. This heuristic was derived from "conventional wisdom" in creating this specific power supply. A3 has a heat sink since it may dissipate approximately 3 watts of power (due to 8.4 V * .800 A = 2.7 Watts). The circuit as a whole does not draw that much power but the heat sink will ensure a long life for A3. A4 is a high frequency sink which tries to remove as much of the ripple from the 5 volt output supply of A3. A5 is present to act as a current sink to drain the power supply when the system is turned off or partially connected (as in when building or testing the device). A6 and A7 constitute the power light of the circuit. A7 has a voltage drop of 1.7 V and we would like a current of about 15mA going through the LED, so (5 V – 1.7V) / (15mA / 1000) = 220 Ohms for A6.

IR Emitter

Figure 2 details the schematic of the infrared emitter circuit. B1 was chosen to allow about 90mA (dissipating ~.5 Watt through B1) of current through B2 and in the actual implementation is comprised of three 10 ohm 1W resistors in series. This implementation detail is due to the availability of parts on hand. The PIC16F688 can only source a current of 20mA through a single pin so since B2 needs more current B4 was needed. Transistor B4 is switched on and off by the PIC16F688 via pin RA4 with logic 0 being off for B4 and logic 1 being on with respect to B4. B3 is chosen as to allow approximately .5mA current

through the base-emitter path of the transistor. B3 protects the transistor from destruction and allows it to enter saturation, acting like a switch and allowing it to supply current through B2.

IR Detector

Figure 3 is the schematic for the infrared detector portion of the circuit. C1 limits the current sunk by RA2 to 5mA since the PIC16F688 cannot sink more than 20mA per pin. Photodiode C2 acts as a short to ground in the presence of infrared radiation and as an open circuit when no infrared radiation is present. Note that output voltage to RA2 does not go to zero in the presence of a full infrared beam, instead being ~.6V due to the voltage drop of C2. This is compensated by the calibration algorithm in the programming for the PIC16F688.

Beam Status Light

Figure 4 is the schematic for the beam status light. This light represents the occlusion or presence of the IR beam. It is on of the IR beam is present and off if the IR beam is occluded. We choose 15mA for the current through D2 (whose voltage drop is 1.7V) so D1 is chosen with (5V – 1.7V) / (15mA / 1000) = 220 Ohms. Since the PIC16F688 may source up to 20mA from a pin, the beam status light is driven directly off of the RA4 pin.

IC Power and Serial Communication

Figure 5 is the schematic for the power supply for the PIC16F688 and MAX233A integrated circuits and the serial port communication path. E5 and E6 sink high frequency signals to ground as to minimize the number of glitches the ICs might experience. E3 is present to prevent current droops to E2. E4 has been hardwired to automatically respond in the affirmative according to peripheral hardware handshaking protocols the host computer might perform. Both the transmission and reception communication channels are present even though the current design only uses the transmission line. This is for future expandability in the form of the host computer being able to speak to the device in case of programming changes on the PIC16F688. E2 is needed to transform the TTL logic the PIC16F688 outputs to the serial line level, which has -12V representing logic one and +12 volts representing logic 0. In the construction of the device, the serial lines were twisted in order to minimize magnetic interference with nearby wires. The serial protocol configuration is 19.2K baud, 8 bit, 1 start bit, 1 stop bit, and no parity bit. The protocol the device emits to the host computer is detailed in Appendix C.

Reset Switch

Figure 6 details the reset switch circuit. It is an RC network where F1 and F2 are tuned to 100 ms (.1 uF * 1000 Ohms = 100ms). A very similar circuit is recommended in the PIC16F688 manual to debounce the reset switch even though the MCLR pin has circuitry to ignore small pulses. F3 is a momentary normally open push button switch and when pushed causes the MCLR line to go to ground halting the PIC16F688. The PIC16F688 is reset when the switch is released and MCLR is again allowed to reach 5 volts.

Project Box and Face Plate

Figure 7 is the layout for the face plate in the project box. The dotted box represents the PCB which is connected inside the box and upside down via 10mm standoffs directly to the faceplate. The 7805 box represents where the 7805 is located with respect to the face plate. This is important since ventilation holes in the side and bottom of the project box are drilled nearby the 7805's heat sink and when the face plate is screwed onto the box the 7805 must be near the holes. The detector and emitter jacks are 1/8 inch mono jacks with a positive tip and negative shield. The detector jack is the left jack, and the emitter jack is the right jack. **The black lead must plug into the detector jack and the red lead must plug into the emitter jack**. The power jack is a size "M" coaxial jack which accepts + tip and – shield power plug. The power switch is labeled with 0 for off and 1 for on. The power light is green and the beam status light is red. The DB25 connector must be female. The project box is 6x4x2 inches in volume and made of a hard black plastic material for easy drilling and shaping. All source/sink wires from the PCB's header to any component on the faceplate must be twisted to minimize magnetic interference.

Instrument Leads

There are two leads for the instrument, the emitter lead and the detector lead. **The emitter lead is the red jacketed lead** and the 1/8 connector tip is positive and the shield is negative. The lead has a shielded cable, but the shield is currently not connected. **The detector lead is the black jacketed lead** whose 1/8 connector tip is positive and shield is negative. This lead also has a shielded cable, but that is also not connected.

PIC16F688 Assembly Code

The assembly code is heavily commented and attached as Appendix D. The only thing of special note is that currently the device simply sends the event changes to the host computer who then performs the timing according to when the host computer received the data. If the device's accuracy is to be increased, then the assembly program for the PIC16F688 would have to be altered so the device performs the timing itself.

Host Computer Program

This simple program is heavily commented and attached as Appendix E. This program simply reads the event data from the device, figures out the time differential from the last event, and prints it out to the screen. There is no special requirement which causes this code to be written to the Linux environment only that it was what my available resources were.

Verification of the Device

Mathematics, detailed in Appendix F, shows the reasoning to detect if the device is working or not. The basic idea is to let a ½ inch diameter rod fall from a known height of ten centimeters through the beam. We calculate the time differential from when the leading edge of the ½ inch diameter rod breaks the beam and when the following edge restores the continuity of the beam. Then the physical system is constructed and we allow the device to measure the actual time difference. Appendix G is the data table and analysis which shows the accuracy of the device and host computer working in tandem.

Recommended Use of Device

The emitter IRLED and the detector photodiode should be enclosed in a blackened shroud, especially the back of the detector. This will prevent ambient IR from causing calibration or beam state detection failures. The emitter and detector should be **securely** attached to their mounting to prevent alignment shifts during use of the device.

Do not drop the detector as it may unseat the ICs inside the enclosure. If they happen to unseat, replace them with an IC insertion tool such that the notch end of the IC matches the notch end of the IC holder and the pin counts match.

Before each measurement run with the device, it is recommended that the device be reset as to recalibrate itself according to the ambient IR conditions.

While precautions in the construction of this device were performed, the device might perform unreliably in the event of strong electromagnetic interference.

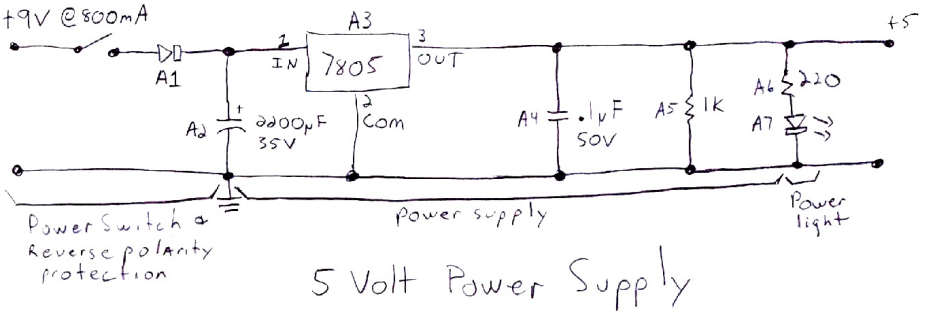
The host operating system of the computer to which the device connects may be any operating system which provides a suitable serial port abstraction.

Do not expose the device to temperatures below 0 degrees centigrade and above 40 degrees centigrade.

If the calibration fails and both on and off values are high, it means that the beam is misaligned.

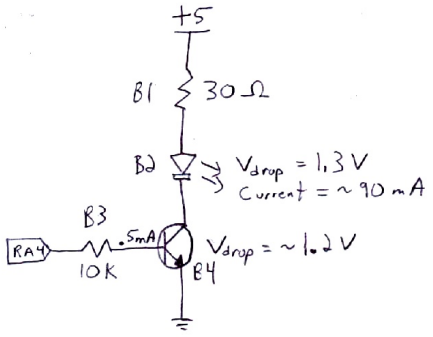
If the calibration fails and both on and off values are low, it means there is too much ambient IR in the room for the device to function.

Figure 1



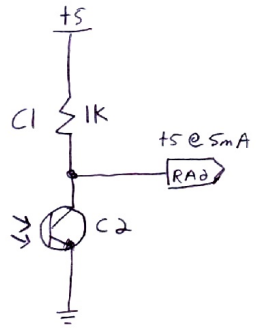
5 Volt Power Supply

Figure 2



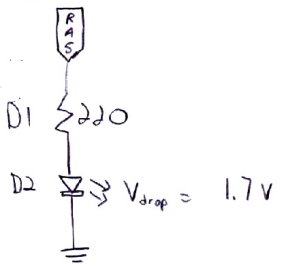
IR Emitter

Figure 3



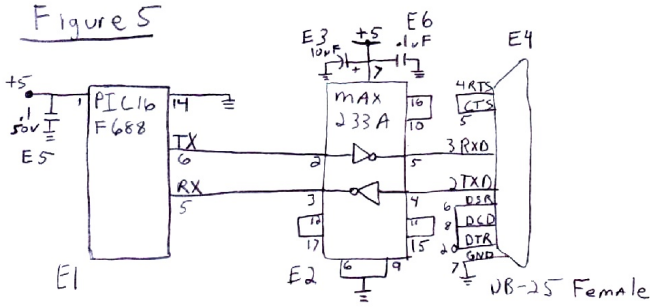
IR Detector

Figure 4



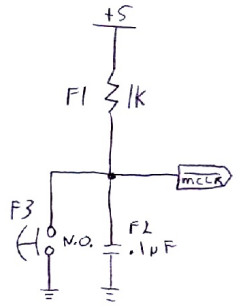
Beam Status Light

Figure 5



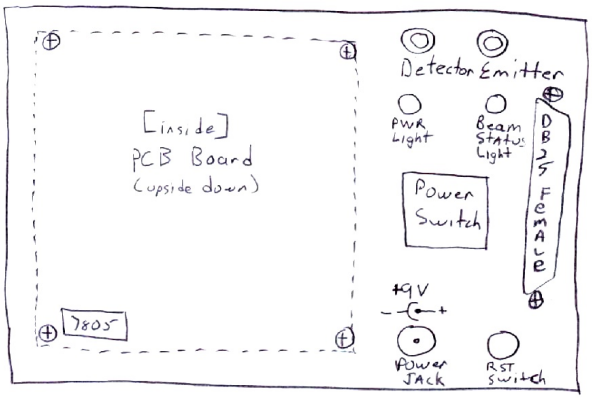
IC Power And Serial Port Connection

Figure 6



Reset Switch

Figure 7



Face Plate Layout

Acknowledgements

I would like to thank Sean Petzold for granting me a wonderful opportunity to build a fun project, Dan Bradley, a friend of mine who checked the mathematical reasoning for the validity test, and last but not least, my wife Stephanie Keller, without whom I would not have been able to complete this project.

Licensing

APPENDIX A

Bill of Materials

Note: All LED packages are 5mm T-1 ¾.

Quantity	Item	Price
1	PIC16F688	$0.00
1	MAX233A	$0.00
2	220 Ohm, 1/4 W	$0.26
3	1 KOhm, 1/4 W	$0.26
1	10 KOhm, 1/4 W	$0.26
3	10 Ohm, 1 W	$2.24
1	LM7805 Voltage Regulator	$1.59
1	1N4004 Diode	$0.19
1	2N2222 NPN Transistor	$0.23
1	10 uF 50V Electrolytic 20%	$0.22
1	2200 uF 35V Electrolytic 20%	$4.49
1	IRLED 1.3V 150mA 950nm	$1.75
1	IR Detector 5Vec 50mA 850nm	$1.75
4	.1uF 50V Ceramic 20%	$2.98
1	14-pin IC Holder	$1.29
1	20-pin IC Holder	$0.69
1	6x4x2 plastic project box	$4.99
2	1/8 inch mono jacks	$3.99
2	1/8 inch mono plugs	$1.50
1	DPDT Power Switch	$3.99
1	Red LED 1.7V, 15mA	$1.00
1	Green LED 1.7V, 15mA	$1.00
1	DB-25 Female with solder leads	$1.99
1	Type M coaxial power jack	$1.99
1	Type M coaxial power plug	$0.00
1	9V DC 800mA Wall Wart	$18.99
1	db-25/db-9 serial cable	$9.99
1	Small Pushbutton N.O. Momentary Switch	$0.87
6 feet	2 conductor twisted/shielded cable	$1.20
Various	Heat shrink tubing small/medium diameter	$1.50
4	10mm standoff with 2 2-3x5mm screws	$2.99
1	TO-220 heat sink	$1.49
1	TO-220 heat sink mounting hardware	$1.99
1	Component PC Board (RS 276-168)	$3.49
Enough	Red 22awg wire	$0.50
Enough	Green 22awg wire	$0.50
Enough	Black 22awg wire	$0.50
4	Cushion Feet	$0.66
2	LED snap holders	$0.60
	Total:	$83.92

APPENDIX B

Construction

One thing to note here is that the PCB board I used from radio shack (detailed in the BOM) has an erroneous labeling of the row numbers, so be aware of this when following these instructions.

The Header Pin Table describes the meaning of each header pin on the PCB. When connecting the header from the PCB, choose the "bottom" holes when soldering them into place. The "upper" holes will be used for connecting the header to the components on the face plate.

HEADER PIN	ASSIGNMENT
1	+9V
2	GND
3	+ Power Light
4	- Power Light
5	Serial TX
6	Serial RX
7	+ IRLED
8	- IRLED
9	+ Beam Status Light
10	- Beam Status Light
11	+ Reset Switch
12	- Reset Switch
13	Not Connected
14	+ Detector Input
15	- Detector Input

The PCB Placement Table describes where each piece is placed onto the specified PC Board. Note that the Serial TX/RX lines must be twisted both from the PIC16F688 to the MAX233A and also from the MAX233A to the header connection. Before attaching the LM7805, ensure to connect the heat sink such that when the LM7805 is placed onto the PC board, the tines of the heat sink point to the inside of the board.

PCB PLACEMENT TABLE

Part	Placement
220 Ohm, 1/4 W	3,B3 - 7,B3
1 KOhm, 1/4 W	9,E4 - 13,E4
1 KOhm, 1/4 W	2,D2 - 6,D2
1 KOhm, 1/4 W	24,B3 - 28,B3
10 KOhm, 1/4 W	3,C4 - 7,C4
10 Ohm, 1 W	5,C2 - 12,C2
10 Ohm, 1W	13,C2 - 20,C2
10 Ohm, 1W	21,C2 - 28,C2
220 Ohm, 1/4 W	8,B3 - 12,B3
1N4004	(cathode) 2,A4 - 2,B3 (anode)
2N2222	C[2,C3] B[3,C3] E[4,C4]

14-pin IC Holder	Pin 1: 10,D4
20-pin IC Holder	Pin 1: 19,D4
10uF 50V Electrolytic	(plus) 27,C5 - 29,D1 (minus)
.1uF 50V Ceramic	9,D3 - 10,D3
.1uF 50V Ceramic	24,D3 - 25,D3
.1uF 50V Ceramic	7,D2 - 8,D2
.1uF 50V Ceramic	22,B3 - 23,B3
2200uF 35V Electrolytic	(plus) 2,A3 - 18,A3 (minus)
LM7805	IN[26,A2] GND[25,A2] OUT[24,A2]
Jumper Power Supply	H1 - 2,B4
Jumper Power Supply	2,A2 - 26,A3
Jumper Power Supply	H2 - 18,A2
Jumper Power Supply	18,A4 - 25,A3
Jumper Power Supply	24,A3 - 22,B2
Jumper Power Supply	25,A4 - 23,B2
Jumper Power Supply	22,B4 - 24,B2
Jumper Power Supply	23,B4 - 28,B2
Jumper Power Supply	24,B4 - 24,B5 (+5V rail)
Jumper Power Supply	28,B4 - 28,C1 (GND rail)
Jumper Power Light	7,B5 - 7,B4
Jumper Power Light	3,B4 - H3
Jumper Power Light	H4 - 1,B3
Jumper PIC Supply	10,D2 - 10,C5
Jumper PIC Supply	10,E3 - 9,D4
Jumper PIC Supply	9,D2 - 9,D1
Jumper MAX233A Supply	25,D2 - 25,C5
Jumper MAX233A Supply	28,D3 - 23,E3
Jumper MAX233A Supply	28,E3 - 24,E3
Jumper MAX233A Supply	27,E3 - 22,E3
Jumper MAX233A Supply	24,D2 - 24,D1
Jumper MAX233A Supply	27,D2 - 27,D1
Jumper IR Emitter	29,B5 - 28,C3
Jumper IR Emitter	21,C3 - 20,C3
Jumper IR Emitter	18,C3 - 17,C3
Jumper IR Emitter	5,C3 - H7
Jumper IR Emitter	H8 - 2,C2
Jumper IR Emitter	4,C2 - 4,C1
Jumper IR Emitter	12,D3 - 7,C3
Jumper Beam Status Light	11,D3 - 12,B4
Jumper Beam Status Light	8,B4 - H9
Jumper Beam Status Light	H10 - 1,C5
Jumper IR Detector	13,E3 - H14
Jumper IR Detector	H15 - 1,E1
Jumper IR Detector	9,E3 - 9,F1
Jumper Reset Switch	2,D3 - 2,D5
Jumper Reset Switch	7,D3 - 6,D3
Jumper Reset Switch	8,D3 - 8,E1
Jumper Reset Switch	7,D4 - 13,D3
Jumper Reset Switch	6,D4 - H11
Jumper Reset Switch	H12 - 1,D3

Jumper Serial Data Path	15,D3 - 20,D3
Jumper Serial Data Path	14,D3 - 21,D3
Jumper Serial Data Path	22,D3 - H5
Jumper Serial Data Path	23,D3 - H6

Once the PCB is finished being populated, you must now drill the holes and mount the face plate components as in Figure 7. When that is finished, then connect the PCB header to the components according to this table. Be sure to twist **every** pair of wires coming from the PCB to a component, and ensure the detector signal lines are far away from anything else, and that the serial signal lines are also far away from anything else. This will help prevent crosstalk, especially for the detector signal lines.

Component Connection Table

Part	Connection
Power Jack	(plus) TIP - DPDT Switch (in)
Power Jack	(minus) Shield - H2
DPDT Switch	DPDT Switch (out) - H1
DB-25	4 - 5
DB-25	6 - 8 - 20
DB-25	7 - Power Jack Shield
DB-25	3 - H6
DB-25	2 - H5
Power Light LED	H3 - Anode
Power Light LED	H4 - Cathode
IR Emitter	H7 - Anode
IR Emitter	H8 - Cathode
Beam Status Light	H9 - Anode
Beam Status Light	H10 - Cathode
Reset Switch	H11 - Reset Switch
Reset Switch	H12 - Reset Switch
IR Detector	H14 - Anode
IR Detector	H15 - Cathode

Constructing the IR Emitter Lead

Take three feet of the two conductor twisted/shielded wire and prepare both ends for soldering. Connect one end to 1/8 inch red plug being careful when soldering and connect the other to the IR LED itself. Be **very** sure to honor the anode and cathode requirements. Seal all connections with heat shrink tubing. For this circuit, the shield is currently ignored.

Constructing the IR Detector Lead

Take three feet of the two conductor twisted/shielded wire and prepare both ends for soldering. Connect one end to 1/8 inch black plug being careful when soldering and connect the other to the photodiode itself. Be **very** sure to honor the anode and cathode requirements. Seal all connections with heat shrink tubing. For this circuit, the shield is currently ignored.

APPENDIX C

Serial Protocol

The device is configured to use 19200 baud with 8-bit data, 1 start bit, 1 stop bit, and no parity bits. With the current programming, the device only sends data out and does not accept any incoming data.

When the device is powered on, it performs a calibration phase and sends this calibration information to the host computer.

If the calibration is successful, then this 7-byte packet of data is sent:

Byte	Value
1	0x43 – C
2	0x4f – O
3	MSB 8 bits of ADC when beam is turned on
4	0x46 – F
5	MSB 8 bits of ADC when beam is turned off
6	0x41 – A
7	Average of byte 3 and 5

If the calibration is not successful, then this 7-byte packet of data is sent:

Byte	Value
1	0x4e - N
2	0x4f - O
3	MSB 8 bits of ADC when beam is turned on
4	0x46 - F
5	MSB 8 bits of ADC when beam is turned off
6	0x41 - A
7	Average of byte 3 and 5

If the device fails to calibrate, it will immediately retry the calibration phase again. This means the host computer will see multiple failed calibration packets in a row until conditions arise that allow a good calibration.

Once the device has successfully calibrated, the implicit state of the beam is that it is in an unbroken state.

If the device detects a beam occlusion, then it sends a “break” event:

Byte	Value
1	0x42 - B

If the device detects that the beam integrity has been restored, it sends a “resume” event:

Byte	Value
1	0x52 - R

Appendix D

PIC16F688 Source Code

```
; This file implements the IR timer code base for Sean Petzold.
; Copyright 2007 Peter K. Keller (psilord@cs.wisc.edu)

; Using the 4MHz internal oscillator means each instruction takes 1 microsecond,
; except for gotos and other branching instructions, which take two.

    list        p=16f688        ; list directive to define processor
    #include <P16F688.inc>        ; processor specific variable definitions
    errorlevel    -302

    __CONFIG    _CP_OFF & _CPD_OFF & _BOD_OFF & _PWRTE_ON &
_WDT_OFF & _INTRC_OSC_NOCLKOUT & _MCLRE_ON & _FCMEN_OFF &
_IESO_OFF


; VARIABLE DEFINITIONS
w_temp      EQU  0x71          ; variable used for context saving
status_temp EQU  0x72          ; variable used for context saving
pclath_temp    EQU  0x73          ; variable used for context saving

spbrg_val    EQU  d'12'         ; 19.2k baud
;spbrg_val   EQU  d'25'         ; 9600 baud
;spbrg_val   EQU  d'207'        ; 1200 baud

; Bit values for beam_status
BROKEN       EQU  0       ; is the beam currently broken, set of true
PREVIOUSEQU  1       ; the previous beam state, set if previously broken

; Bit values for irled_status
ACTIVE    EQU  0       ; set if IRLED is on


; User defined variables and their needed sizes
    cblock 0x20
dvar:1                ; used for Delay Loop
dvar2:1               ; used for Delay Loop
tmp0:1
tmp1:1
adc_hi:1         ; storage for ADC high byte
adc_lo:1         ; storage for ADC low byte
irled_status:1       ; 0 for off, 1 for on
beam_status:1      ; various bit values for things about the beam
calib_on:1           ; MSB 8-bits of the calibration ADC value for irled on
```

```
calib_off:1         ; MSB 8-bits of the calibration ADC value for irled off
calib_val:1         ; calib value which, when an adc sample is compared to it,
                    ; it denotes the beam is present or broken.
    endc

;*****************************************************************
    ORG     0x000           ; processor reset vector
    goto  main              ; go to beginning of program

    ORG     0x004           ; interrupt vector location
    movwf   w_temp          ; save off current W register contents
    ORG     0x005           ; part of the interrupt vector
    movf STATUS,w           ; move status register into W register
    movwf   status_temp     ; save off contents of STATUS register
    movf PCLATH,w     ; move pclath register into W register
    movwf   pclath_temp     ; save off contents of PCLATH register

; for now, just ignore any interrupts we get

    movf pclath_temp,w      ; retrieve copy of PCLATH register
    movwf   PCLATH          ; restore pre-isr PCLATH register contents
    movf status_temp,w      ; retrieve copy of STATUS register
    movwf   STATUS          ; restore pre-isr STATUS register contents
    swapfw_temp,f
    swapfw_temp,w           ; restore pre-isr W register contents
    retfie                  ; return from interrupt


main:
    bcf     STATUS, RP0 ; Bank 0
    call    InitVariables       ; set up the state of various variables
    call    InitIO              ; set up all of my digital/analog pins and init the ADC
    call    CalibrateIRBeam     ; Try over and over to calibrate the beam, telling the
                                ; PC what it is doing.
Loop
    call    DebounceBeamState
    call    IsBeamBroken
    call    NotifyBeamChange
    call    DisplayADCResult
    goto    Loop                ; infinite loop

; A simple delay function for ~1.4 milliseconds
Delay_1ms:
    movlw   0x01      ; set outer delay loop
    movwf   dvar2
Delay0
```

```
    movlw   0xFF
    movwf   dvar       ; set inner delay loop
Delay1
    decfsz    dvar, F
    goto   Delay1

    decfsz    dvar2, F
    goto   Delay0

    return

; Initialize various variables.
InitVariables
    bcf        STATUS, RP0
    clrf    beam_status
    bcf        beam_status, BROKEN      ; the beam is not broken
    bcf        beam_status, PREVIOUS   ; the previous state is not broken either.
    return

; Initialize the IO pins to be what I need them to be
InitIO:
    bcf        STATUS, RP0     ; select bank 0
    movlw   0x07                    ; turn off the comparators leaving the pins digital
    movwf   CMCON0
    call    InitPortA
    call    InitPortC
    call    InitSerial
    call    InitADC
    bcf        STATUS, RP0     ; back to bank 0
    return

; For this function, we initialize PortA to have RA2/AN2 be configured as an
; analog input for ADC. RA3 is also a *mandatory* digital input pin.
InitPortA:
    bcf        STATUS, RP0     ; Bank 0 for PORTA register
    clrf    PORTA                   ; mark all lines as zero
    bsf        STATUS, RP0     ; Bank 1 for ANSEL/TRISA registers
    movlw   b'00000100'        ; ansel<2> is marked as analog input
    movwf   ANSEL
    movlw   b'00001100'        ; ra3, ra2 are inputs. ra2 is now an analog input
    movwf   TRISA
    return

; For this function, all pins of PortC are configured as digital except rc5, which
; will be the serial rx port.
InitPortC:
```

```
    bcf     STATUS, RP0 ; Bank 0 for PORTC reg
    clrf    PORTC       ; mark all lines as zero
    bsf     STATUS, RP0 ; Bank 1 for TRISC reg
    movlw   b'00110000' ; rc5:4 tristate for serial port, rest are outputs
    movwf   TRISC
    bcf     STATUS, RP0 ; Back to bank 0
    return


; This function sets up and turns on the ADC. It assumes the InitPortX functions
; have already set up the correct inputs as analog
InitADC:
    bsf     STATUS, RP0 ; Bank 1
    movlw   b'0001000'  ; set ADC Fosc/8 clock
    movwf   ADCON1
    bcf     STATUS, RP0 ; Bank 0
    movlw   b'0001001'  ; left justified, Vdd, AN2, not in progress, enabled
    movwf   ADCON0
    call    ADCDelay    ; wait for the ADC to stabalize
    return

; set up the serial port for communications...
InitSerial:
    bcf     STATUS, RP0 ; Bank 0

    ; set up the baud rate
    movlw   spbrg_val   ; the baud I'm going to use
    movwf   SPBRG

    ; set up the transmit & control register
    bcf     STATUS, RP0 ; Bank 0
    movlw   b'00100110' ; don't care, 8-bit, xmit enable, async,
                        ; sync break xmit compl., high speed, TSR empty, 9th bit
    movwf   TXSTA

    ; set up the receive & control register
    bcf     STATUS, RP0 ; Bank 0
    movlw   b'10010000' ; serial port enable, 8 bit, don't care, en recv
                        ; don't care, no frame err, no overrun err, no 9th bit
    movwf   RCSTA
    call    Delay_1ms   ; make sure it stabalizes
    movf RCREG, W       ; clean out the usart recv buffer
    bcf     RCSTA, CREN ; clear any errors if there were any
    bsf     RCSTA, CREN
    return
```

```
; This function does a polling sample of the ADC, then leaves the values in
; adc_hi and adc_lo.
SampleADC:
    bcf     STATUS, RP0 ; Bank 0
    bsf     ADCON0, GO  ; Start the acquisition

    btfsc  ADCON0, GO     ; Poll for completion
    goto   $-1
                          ; Grab the result out
    bcf     STATUS, RP0 ; Bank 0
    movf ADRESH, W        ; Grab the high part
    movwf   adc_hi
    bsf     STATUS, RP0 ; Bank 1
    movf ADRESL, W        ; Grab the low part
    movwf   adc_lo

    return

; This function assumes that the ADC has just been sampled. It sets beam_status
; to zero if, according to the ADC the beam has not been broken, and one if
; according to the ADC, the beam has been broken.
IsBeamBroken:
    bcf     STATUS, RP0         ; Bank 0
    bcf     beam_status, BROKEN     ; assume no it isn't
    movf adc_hi, W
    subwfcalib_val, W               ; Perform: calib_val - W
                                    ; If W is eq/smaller than calib_val, then
                                    ; beam is *not* broken.
                                    ; C is one if W <= calib_val
    btfss  STATUS, C                ; is C zero (so W > calib_val)
    bsf     beam_status, BROKEN     ; yes, set broken bit

    return

; This function samples the ADC with a delay in the middle and only if both
; samples agree is the beam considered stable
DebounceBeamState:
    bcf     STATUS, RP0     ; bank 0
    movf beam_status, W         ; save previous beam status
    movwf   tmp0
    call    SampleADC           ; get the first reading
    call    IsBeamBroken        ; check it
    movf beam_status, W
    andlw1<<BROKEN          ; look only at the broken/present bit
    movwf   tmp1                ; save the beam state
    call    Delay_1ms           ; wait ~1 millisecond
```

```
    call    SampleADC           ; get the second reading
    call    IsBeamBroken        ; check it
    movf beam_status, W
    andlw1<<BROKEN      ; look only at the broken/present bit
    subwftmp1, W            ; check beam stability
    btfsc  STATUS, Z        ; Is Z one (for the borrow)?
                                ; 0 - 0 = 0; 1 - 1 = 0
    goto   DBS_debounced        ; yup, state is debounced
    movf tmp0, W                ; nope, so restore original beam state
    movwf   beam_status
    return
DBS_debounced
    movf tmp0, W                ; grab old beam state
    movwf   beam_status
    andlw1<<PREVIOUS    ; keep whatever the previous bit was
    iorwf tmp1, W               ; mix it with new beam status
    movwf   beam_status         ; fix it up nice for IsBeamBroken
    return

; Wait a minimum of 6 microseconds for the ADC to stabilize
ADCDelay:
    nop
    nop
    nop
    nop
    nop
    nop
    return

; This function performs some averaging for an ADC sample when the beam is off
; and when the beam is on. This will give us voltage brackets which dictate
; the state of the beam given ambient IR conditions. If the beam cannot be
; calibrated, info concerning such will be output to the serial port and the
; calibration retried. Notice that when the irled is off, the ADC will read a high
; value due to the circuit design. When the irled is on, a low adc value will be
; read.
CalibrateIRBeam:
    bcf     STATUS, RP0 ; Bank 0
    movlw   b'00000000'     ; turn irled off (RA4)
    movwf   PORTA
    call    Delay_1ms           ; Wait for transistor & photodiode to stabilize
    call    SampleADC           ; grab a sample
    bcf     STATUS, RP0 ; Bank 0
    movf adc_hi, W
    movwf   calib_off       ; store it in the irled off calibration value, the
                                ; voltage, and therefore the bit pattern is high
```

```
                                ; in the case of an off irled

movlw   b'00010000'     ; turn beam on (RA4)
movwf   PORTA
call    Delay_1ms               ; wait for the transistor & photodiode to stabilize
call    SampleADC               ; grab a sample
bcf     STATUS, RP0 ; Bank 0
movf adc_hi, W
movwf   calib_on                ; store it in the irled on calibration value, the
                                ; voltage, and therefore the bit pattern is low
                                ; in the case of an on irled

; make sure the irled is left turned on for the rest of the execution
clrf    irled_status
incf    irled_status, F

; now perform the average of the two to get a "zero" that when the ADC sample
; crosses, it means the beam is on or off.
; This next piece of code implements an average as:
; calib_on/2 + calib_off/2 = calib_val

; calib_on/2
movf calib_on, W
movwf   tmp0
bcf     STATUS, C
rrf     tmp0, F

; calib_off/2
movf calib_off, W
movwf   tmp1
bcf     STATUS, C
rrf     tmp1, F

; Add them (this will fit into an 8 bit value)
bcf     STATUS, C       ; Get rid of polluted carry bit
movf tmp0, W
addwf   tmp1, W         ; complete the average

; store the calibrated value for later use
movwf   calib_val

; check the validity of the calibration data and retry if bad.
; This is done by subtracting the low from the high and seeing
; if there is enough "room" to have a good reading. Room is
; defined as the result of the subtraction being more than 25%
; of the total range. This implies a %12.5 range in the "on/off"
```

```
    ; region around the average, which is only decimal number 13, a tight
    ; but meaningful range. In this case it would be 256 * .25, which is
    ; ~64. So, if the subtraction results in a number less than 64, we
    ; know the calibration probably doesn't make sense
    movf calib_on, W
    subwfcalib_off, W
    btfss  STATUS, C        ; was calib_on(low voltage) > calib_off(high voltage)?
    goto   CIRB_retry       ; yes, retry
    movwf   tmp0            ; nope, so do range check
    ; check it against 64
    movlw   d'64'
    subwftmp0, W
    btfss  STATUS, C        ; did the subtraction result in a borrow?
    goto   CIRB_retry       ; yes, so retry
    call   SendGoodCalib    ; nope, so there is enough room so use the calibration
    call   SendCalibData
    return

CIRB_retry                  ; we lose, tell the pc what happened and try again.
    call   SendBadCalib
    call   SendCalibData
    call   Delay_1ms        ; let things change a bit before retrying
    goto   main             ; retry

    return


; This function displays the ADC highest 4-bits, the pin format is:
; MSB RC3 RC2 RC1 RC0 LSB
; RA5 is beam broken/unbroken light
; RA4 is IR LED on/off
DisplayADCResult:
    bcf        STATUS, RP0     ; Bank 0

    clrw

    ; now, we ior in the beam broken/present bit
    iorlw b'00100000'               ; assume beam is unbroken, turn on ra5
    btfsc  beam_status, BROKEN      ; is the beam broken?
    andlwb'11011111'            ; yes, then turn off ra5


    ; Or in whether or not the IR emittor is turned on or not
    ; since the pin which controls the IR LED is also on this port.
    btfsc  irled_status, ACTIVE
    iorlw b'00010000'     ; turn on RA4 if irled is turned on
```

```
    movwf   PORTA           ; Shove it out to PORTA<1:0>

    ; Display the most significant 4 bits
    swapfadc_hi, W      ; shove the high nibble into the low nibble spot
    andlw0x0F           ; Only want the LSB 4 bits
    movwf   PORTC       ; Shove it out to PORTC<3:0>

    return


; If the calibration succeeded, then this must appear before the calibration data
; is written.
SendGoodCalib:
    movlw   0x43                ; C
    call    TransmitSerial      ; Send the fact I calibrated successfully
    return

; If, for whatever reason, the calibration, failed, this tells the PC about it.
SendBadCalib:
    movlw   0x4E                ; N
    call    TransmitSerial      ; Send the fact I did not calibrate successfully
    return

; Send a packet of calibration data (good or bad) to the PC
SendCalibData:
    movlw   0x4F                ; O for "on"
    call    TransmitSerial
    bcf     STATUS, RP0 ; bank 0
    movf calib_on, W        ; The on calibration byte
    call    TransmitSerial
    movlw   0x46                ; F for "off"
    call    TransmitSerial
    bcf     STATUS, RP0 ; bank 0
    movf calib_off, W           ; The off calibration byte
    call    TransmitSerial
    movlw   0x41                ; A for "average"
    call    TransmitSerial
    bcf     STATUS, RP0 ; bank 0
    movf calib_val, W           ; The calibration value
    call    TransmitSerial
    return

SendBeamResumeEvent:
    movlw   0x52                ; R for beam restore event
    call    TransmitSerial
```

```
    return

SendBeamBreakEvent:
    movlw   0x42                ; B for beam break event
    call    TransmitSerial
    return


NotifyBeamChange:
    btfsc  beam_status, BROKEN      ; is beam broken?
    goto   NBC_is_prev_broken       ; yes, check if previous was broken
    goto   NBC_is_prev_resume       ; nope, check if previous was present

NBC_is_prev_broken
    btfsc  beam_status, PREVIOUS    ; was the beam previously broken?
    return                          ; yes, no event happened
    goto   NBC_beam_break_event     ; nope, so a break event happened

NBC_is_prev_resume
    btfsc  beam_status, PREVIOUS    ; was the beam previously broken?
    goto   NBC_beam_resume_event    ; yes, so a resume event happend
    return                          ; no, so nothing happened

NBC_beam_break_event
    call    SendBeamBreakEvent
    bsf     beam_status, PREVIOUS   ; make previous match current
    return

NBC_beam_resume_event
    call    SendBeamResumeEvent
    bcf     beam_status, PREVIOUS   ; make previous match current
    return


; ------------------------------------------------------------------------------

ReceiveSerial:
    bcf     STATUS, RP0     ; bank 0
    btfss  PIR1, RCIF           ; check if data
    return                      ; if none then return
    btfsc  RCSTA, OERR          ; if overrun happened
    goto   ErrSerialOverr
    btfsc  RCSTA, FERR          ; if framing error happend
    goto   ErrSerialFrame
    movf RCREG, W
    return
```

```
ErrSerialOverr:
    bcf     RCSTA, CREN     ; clear the error
    bsf     RCSTA, CREN
    return

ErrSerialFrame:
    movf RCREG, W           ; dump it
    return

TransmitSerial:
    bcf     STATUS, RP0     ; bank 0
    btfss  PIR1, TXIF
    goto  $-1
    movwf   TXREG
    return


    ORG 0x2100              ; data EEPROM location
    DT    "Version 1.0"
    DE    0
    DT    "Created by: Peter K. Keller (psilord@cs.wisc.edu)"
    DT    0

    END
```

APPENDIX E

Example Host Computer Client Program

The function of the client program is to suitably configure the host computer's serial port and then read the calibration and beam break/resume events from the device. The provided client program merely computes the time difference between every successive byte the device sends the host computer. This computes the time difference between break and resume events of the beam.

```
/* Copyright 2007 by Peter K. Keller */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <ctype.h>
#include <sys/time.h>

/*
'open_port()' - Open serial port 1.
Returns the file descriptor on success or -1 on error.
*/

int open_port(char *file)
{
    int fd; /* File descriptor for the port */
    struct termios options;

    fd = open(file, O_RDWR | O_NOCTTY | O_NDELAY);

    if (fd == -1) {
        printf("open_port: Unable to open %s %d(%s)\n",
            file, errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    /* In this case, we want it read only */
    if (fcntl(fd, F_SETFL, O_RDONLY) < 0) {
        perror("fcntl -");
        exit(EXIT_FAILURE);
    }
```

```
    if (tcgetattr(fd, &options) < 0) {
        perror("tcsetattr -");
        exit(EXIT_FAILURE);
    }

    /* set baud rate */
    if (cfsetispeed(&options, B19200) < 0) {
        perror("cfsetispeed() - ");
        exit(EXIT_FAILURE);
    }

    if (cfsetospeed(&options, B19200) < 0) {
        perror("cfsetospeed() - ");
        exit(EXIT_FAILURE);
    }

    /* raw in and out */
    cfmakeraw(&options);

    /* enable the receiver and set local mode */
    options.c_cflag |= (CLOCAL | CREAD);

    /* 8-bit, 1 start bit, 1 stop bit, no parity bit */
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;

    /* disable hardware flow control */
    options.c_cflag &= ~CRTSCTS;

    /* disable software flow control too */
    options.c_iflag &= ~(IXON | IXOFF | IXANY);

    if (tcsetattr(fd, TCSANOW, &options) < 0) {
        perror("tcsetattr -");
        exit(EXIT_FAILURE);
    }

    printf("Opened serial port!\n");
    return (fd);
}


/* calculate elapsed time between now and then, now is assumed to be
    more recent than then. It returns the difference in milliseconds. */
```

```
double event_time(struct timeval *now, struct timeval *then)
{
    struct timeval a, b;

    /* structure copy */
    a = *now;
    b = *then;

    /* translate the seconds elapsed into microseconds */
    while(a.tv_sec > b.tv_sec) {
        a.tv_usec += 1000000;
        a.tv_sec--;
    }

    /* calculate microsecond resolution */
    return (a.tv_usec - b.tv_usec) / 1000.0;
}

int main(int argc, char *argv[])
{
    int fd;
    unsigned char read_byte;
    struct timeval timestamp;
    struct timeval previous = {0,0};
    int once = 0;

    if (argc != 2) {
        printf("Usage:\n");
        printf("%s /dev/ttyS[0-9]+\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    printf("Opening serial port: %s\n", argv[1]);
    fd = open_port(argv[1]);
    if (fd < 0) {
        printf("Could not open serial port!\n");
        exit(EXIT_FAILURE);
    }

    printf("Reading data from PIC\n");

    /* while there is data, read it. */
    while (read(fd, &read_byte, 1) > 0) {
        gettimeofday(&timestamp, NULL);

        printf("Read: 0x%02x : '%c' @ %lu sec : %lu usec",
```

```
            read_byte, isprint(read_byte)?read_byte:' ',
            timestamp.tv_sec, timestamp.tv_usec);

        if (once == 0) {
            once = 1;
            previous = timestamp;
            printf("\n");
        } else {
            printf(" : %lf ms\n", event_time(&timestamp, &previous));

            previous = timestamp;
        }
    }
    printf("Done!\n");

    printf("Closed serial port!\n");
    close(fd);

    return 0;
}
```

APPENDIX F

Verification of the Device

This section describes the experiment which will determine the validity of the device. Since the function of the device is to time the event changes of an infrared beam, the experiment will be designed to find an analytical solution to a beam breakage/resumption scenario and then the device will be tested to see how well it conforms to the analytical solution. Figure 8 describes the physical scene of the experiment. The purpose of this setup is to time how long the beam is broken by the rod during its fall.

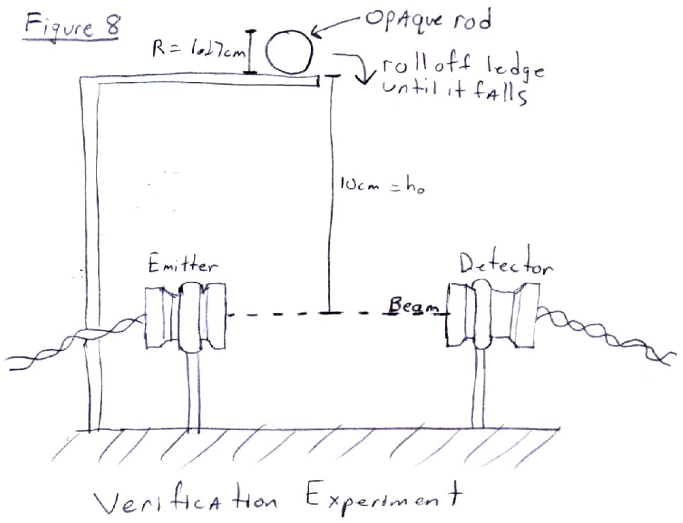


Figure 9 shows the mathematical reasoning which analytically computes how long the beam should be occluded by the ½" (1.27cm) diameter opaque rod as it falls through the beam from a height of 10 cm.

Figure 9

Let $g = -9.8\ m/s^2$

Start with equation for acceleration in time:

$$a(t) = g$$

Integrate it to get velocity:

$$v(t) = gt + C$$

At $t=0$ the initial velocity is also 0:

$$0 = g(0) + C$$

$$0 = C$$

Therefore:

$$v(t) = gt$$

Integrate to get vertical position:

$$y(t) = \frac{g}{2}t^2 + h$$

At $t=0$ the initial height to the bottom part of the Rod is .10m

$$.10m = \frac{g}{2}(0) + h$$

$$.10m = h$$

Therefore:

$$y(t) = \frac{g}{2}t^2 + .10m$$

We Are interested at what t is when $y(t)$ is zero:

Let $h_0 = .10m$

$$0 = \frac{g}{2}t^2 + h_0$$

$$-h_0 = \frac{g}{2}t^2$$

$$\frac{2(-h_0)}{g} = t^2$$

$$\sqrt{\frac{-2(h_0)}{g}} = t_b$$

t_b is when the bottom edge of the rod crosses the beam which interrupts it.

By inspection, we compute t_t, which is when the top of the rod crosses the beam, which restores the integrity of the beam.

$$\sqrt{\frac{-2(h_0+R)}{g}} = t_t$$

So, the time the beam is occluded is:

$$\Delta t = t_t - t_b$$

$$\Delta t = \sqrt{\frac{-2(h_0+R)}{g}} - \sqrt{\frac{-2(h_0)}{g}}$$

If we let R = .0127m and h0 = .10m, then dt = 8.8 msec. So, when we set up the physical experiment, we will see how well the device measures the 8.8 msec occlusion of the beam.

APPENDIX G

Experimental Data for Verification of Device

Drop Trial	Beam Occlusion Time (ms)
1	8
2	8
3	7
4	7
5	8
6	7
7	8
8	10
9	8
10	7
11	8
12	7
13	7
14	8
15	8
16	8
17	8
18	8
19	8
20	7
21	7
22	13
23	8
24	8
25	8
26	7
27	8
28	7
29	8
30	7
31	8
32	8
33	7
34	8
35	8
36	8
37	8
38	7
39	8
40	7
41	8
42	8
43	8
44	8
45	8

46	8
47	9
48	10
49	9
50	10
51	9
52	10
53	9
54	10
55	34
56	10
57	10
58	10
59	8
60	8
61	8
62	8
63	7
64	8
65	8
66	8
67	8
68	8
69	8
70	8
71	8
72	8
73	8
74	8
75	8
76	8
77	48
78	8
79	8
80	8
81	8
82	8
83	8
84	8
85	8
86	8
87	8
88	8
89	8
90	8
91	8
92	7
93	8
94	8
95	8

96	8
97	8
98	7
99	8
100	7
Average	8.73
Standard Deviation	4.82
Analytical Value	8.8
% Error	0.80%

Analysis of Data

Here we see that the device measured pretty closely to the accepted analytically computed value of 8.8ms. The standard deviation is a little higher than expected due to the outliers in the occlusion time data.

Sources of Error

1) The rod was released each time by hand. Through observation, all of the outliers were because the rod hadn't been aligned properly on the test platform before release and had fallen crookedly through the beam causing a bad time read. I believe this specific source of error contributed significantly to the wider than expected standard deviation.

2) If the host computer suddenly became busy during the time the beam was occluded, then since the device itself does not compute the time difference of the beam events, the computer might have been delayed in determining the time differential.

3) If during the use of the device, the emitter and detector shift and become slightly misaligned. This would cause a variation as to when the device thinks the beam is broken or not and would contribute to an error in reading.

Bibliography

Hayes, Thomas C. and Horowitz, Paul. Student Manual for The Art of Electronics 2nd ed.
New York: Cambridge University Press, 1989.

Hayes, Thomas C. and Horowitz, Paul. The Art of Electronics 2nd ed.
New York: Cambridge University Press, 1989.

Katzen, Sid. The Quintessential PIC Microcontroller.
Great Britain: Springer, 2001.

Garbutt, Mike. Asynchronous Communications with the PICmicro USART.
U.S.A.: Microchip Technology Inc, 2003.

Microchip. PIC16f688 Data Sheet 14-pin flash based , 8-bit CMOS Microcontrollers with nanoWatt Technology.
U.S.A: Microchip Technology Inc, 2007

Transistor Circuits. Online. 27 December 2007.
<http://www.kpsec.freeuk.com/trancirc.htm>

Transistor 2N2222 Specifications. Online. 27 December 2007.
<http://www.ciphersbyritter.com/NOISE/SPEC2222.HTM>

Johns, W. E. Notes on LEDs. Online. 27 December 2007.
<http://www.ciphersbyritter.com/NOISE/SPEC2222.HTM>

Voltage Regulators. Online. 27 December 2007.
<http://www.teachers.ash.org.au/jfuller/electronics/regulators.htm>

How long will take for a stone freely dropped from a height of 39.2m to hit the ground?. Online. 27 December 2007.
<http://answers.yahoo.com/question/index?qid=20071103235046AAxZWdZ>

Martel, Mike. Using Transistors as Switches. Online. 27 December 2007.
<http://www.rason.org/Projects/transwit/transwit.htm>

SparkFun Electronics. Online. 27 December 2007.
<http://www.sparkfun.com/commerce/present.php?p=BEE-1-PowerSupply>

Express, Electronix. Electronics Tips: Contact Bounce and De-bouncing. Online. 27 December 2007.
<http://www.elexp.com/t_bounc.htm>

Ganssle, Jack G. A Guide to Debouncing. Online. 27 December 2007.
<http://www.ganssle.com/debouncing.pdf>