A Microcontroller-Based Precision Xenon Strobe System

Tom Wheeler, N0GSG
twheeler@kc.devry.edu
January 2005

WARNING

The circuitry described in this paper produces voltages of more than 350 volts and utilizes potentially lethal energy levels. Do not construct this project without appropriate knowledge and safety procedures. If you don't have the background to do this project safely, get qualified assistance!

Introduction

There are many applications where a precisely-controlled high-intensity xenon strobe system comes in very handy. In photography, a precision strobe will let you accurately capture motion and velocity, and realize effective remote control for "slave" strobe systems (using an IR or RF data link). Strobes can be setup for measuring the timing of events in machinery, and even used as wireless signaling devices (such as in traffic preemption systems).

The circuit described in this paper is a microprocessor-controlled precision strobe that operates from a 10 to 16 V DC supply (requiring less than 2A for typical operation). It uses an Atmel AT90S2313 single-chip microcontroller to develop the timing signal. The '2313 was chosen because it has an onboard UART, which would be very useful for adapting this project to a remote or PC-controlled setup. Only two I/O pins of the microcontroller are used! Timing is derived from a 4.9152 MHz crystal.

Circuit Analysis

Figure 1 shows the entire circuit. There are three primary sections, a DC to DC step-up inverter, the microcontroller, and a trigger circuit for the xenon flash tube.

The inverter circuit is composed of four transistors, Q_1 through Q_4. Q_2 and Q_3 form a push-pull multivibrator with a frequency of about 880 Hz. The outputs at the collectors of Q_2 and Q_3 are approximately square waves with a 180 degree phase difference. The square wave outputs directly drive the gates of power MOSFETs Q_1 and Q_4, resulting in an AC square wave signal driving the primary of T_1, the step-up transformer, which is a standard 12V CT @3A to 120 V unit. The output of T_1 is fed into the voltage doubler, which consists of C_7, C_9, D_2, and D_1, resulting in a resting voltage of about +390 V across C_9. The prototyped inverter was better than 95% efficient -- *no* heatsinks are needed on Q_1 and Q_4 in this application!

The value of C_9 controls the amount of energy delivered to the flash tube for each flash, which can be calculated by:

(1) $$PE = \frac{1}{2} C_9 V_Q^2$$

Where C_9 is the value of the "dump" capacitor, and V_Q is the standing (quiescent) voltage at firing time (about 390 V in this circuit). For safety, you should evaluate the amount of energy being driven into the flashtube you intend to use and make sure that the ratings aren't being exceeded at the flash rate.

The microcontroller is comprised of U_2, the AT90S2313. The MCU is provided regulated 5 volts by U_1, and Y_1 determines its operating frequency. The value of 4.9152 MHz provides optimum choice of data rates for the UART, should a serial interface be implemented for the unit. Port output PB0 provides the trigger signal for the strobe; this signal is current boosted by Q_5 prior to driving the trigger stage. Pin PD0 is used as an external input; the system soft-switches between internal and external trigger by examining this pin during initialization.

On power-up the software examines the state of pin PD0, which is connected to the EXT IN connector through an ESD suppression network (R_{11}, C_{15}) and a pull-up resistor (R_{10}). If this signal measures "1," then the processor simply accepts an input signal from J_1 and passes it out through PB0 to trigger the strobe. If the MCU sees a "0" on PD0 during power-up, it begins firing the flash at a rate determined by the software timing loop (up to 30 seconds total).

The trigger circuit consists of Q_6, C_{11}, R_7, and T_2. Initially, Q_6 is non-conductive (off) and C_{11} charges to at least 300 V through R_7. When a trigger current is applied to the gate of Q_6, it turns on, rapidly dumping the charge from C_{11} into the primary of T_2, which then steps the pulse up to approximately 4 kV. The 4 kV pulse is applied to the trigger electrode of the xenon flash tube; the pulse provides enough ionization of the xenon gas to allow the tube to fully conduct, pulling the energy from C_9. Q_6 turns off as C_{11} discharges ($I_{HOLD\text{-}Q6} < (V_{HV} / R_7)$, and the tube ceases conducting once the anode to cathode voltage falls below about 40 volts, preparing the circuit for another firing cycle. Resistor R_6 helps guarantee lamp commutation (shut-off), but may be omitted to obtain higher speed strobes (the circuit can easily operate over 40 Hz even with R_6 in place). The typical pulse width of the emitted light beam is much less than 500 μs.

Construction

The circuit was built on two printed circuit boards layed out using EAGLE CAD, but could easily be constructed using conventional point-to-point methods. The inverter and microprocessor were placed on one board, and the xenon lamp and high-voltage trigger circuitry were placed on the second board. This was done to allow the circuitry to fit into a desired enclosure (a discarded hand-held spotlight case, which provides a useful reflector and insulation from the high voltage circuitry).

Don't skimp on the high voltage capacitors; this circuit provides sufficient output power (better than 40W, easily) to destroy underrated components in a flash! C_9 should be temperate rated at 100 °C. ***Make absolutely certain that the high voltage components are untouchable by the user***. Note that the ground of the circuit is common between the low and high-voltage sections; there's no particular need for isolation here.

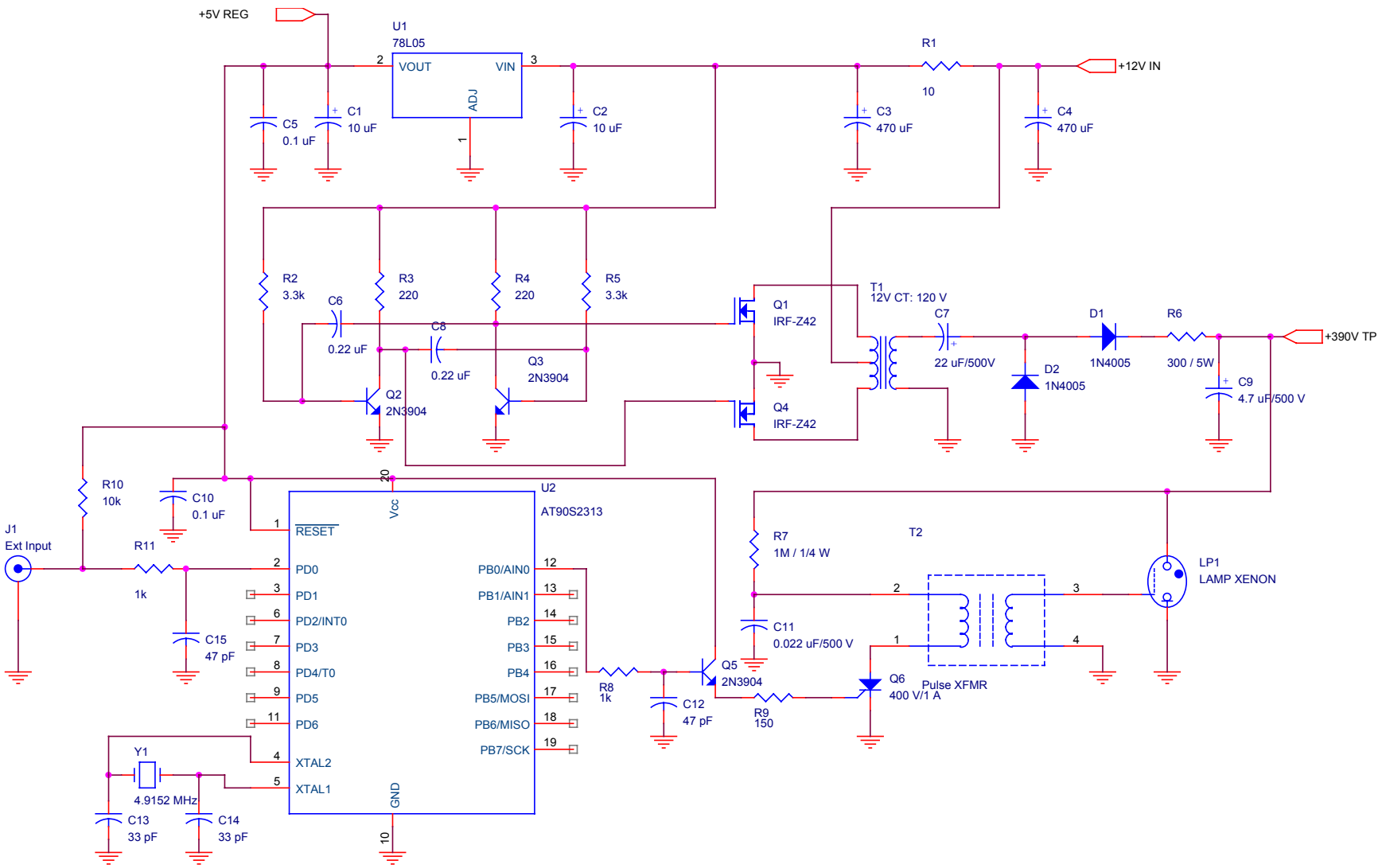

Figure 1: Schematic diagram of the precision strobe system

Software

You'll need software to drive the microcontroller. The prototype software simply flashes the strobe at a predetermined rate for around 25 seconds, then shuts off the preserve the flash tube and high voltage components. By adjusting the constants in the listing, you can obtain very precise flash rates. You will probably want to modify the code for your particular purpose. Because the circuit is based on a crystal oscillator, the frequency will be very precise.

Of course, if you already have a precise frequency source, you can dispense with the microcontroller and directly fire the strobe by triggering the base of Q_5.

Listing 1 shows the code as implemented in the prototype unit. Note that some functions may not be directly called. This is because the software is built around a standardized application framework for this particular processor which has the commonly-needed functions built in.

Conclusion

A precision strobe system need not be expensive. The unit detailed in this article is reliable, efficient, and flexible in its operation. It should be useful in many fields of study.

```
; LISTING 1: strobe.asm
;
; Controller for precision strobe.
;
; Target: AT90S2313 (ATMEL), 4.9152 MHz clock
;
; Version 1.0
;
; Author: Wheeler, T (N0GSG)
;
;
; I/O Pin Descriptions:
;
; PB0: OUTPUT, Strobe pulse to fire xenon tube; fires on L-H transition.
; PB1-PB7: OUTPUTS, Unused
;
; PD0: INPUT. Checked on power up. If (PD0==0), then strobe mode is entered. If
;             (PD0==1), external input mode is entered.
;
; Note: The internal trigger frequency is 19.2582 (+/- 0.001 Hz) in this version.
;
; PD1-PD6: OUTPUT, UNUSED
;
; Register Usage:
;
; R10: Last Switch Status. Holds result from last state change of switch on PD3;
;      used to determine which way to program INT1 edge-sensitivity.
;

.equ    portd=$12
.equ    ddrd=$11
.equ    pind=$10
.equ    portb=$18
.equ    ddrb=$17
.equ    eecr=$1c
.equ    eedr=$1d
.equ    eear=$1e
.equ    pinb=$16
.equ    acsr=$08
.equ    tccr0=$33
.equ    tcnt0=$32
.equ    mcucr=$35
.equ    gimsk=$3b
.equ    timsk=$39
.equ    sreg=$3f
.equ    udr=$0c
.equ    ucr=$0a
.equ    ubrr=$09
.equ    usr=$0b
.equ    tccr1a=$2f
.equ    tccr1b=$2e
.equ    tcnt1h=$2d
.equ    tcnt1l=$2c
.equ    ocr1ah=$2b
.equ    ocr1al=$2a
.equ    icr1h=$25
.equ    icr1l=$24

.equ    spl=$3d
.equ    RAMHIGH=$df
.equ    RAMLOW=$60

.equ    TEMPBUF=$c0
.equ    FRAMEBUF=RAMLOW       ;frame buffer
.equ    INPUTBUF=RAMLOW+80    ;line-input buffer
.equ    MAXDATA=64            ;maximum number of data bytes allowed in a frame
```

```
        .org    0               ;code always begins at address 0
        rjmp    start
        rjmp    ext_int0
        rjmp    ext_int1
        rjmp    timer1_capt1
        rjmp    timer1_comp1
        rjmp    timer1_ovf1
        rjmp    timer0_ovf
        rjmp    uart_rx
        rjmp    uart_dre
        rjmp    uart_tx
        rjmp    ana_comp

;
; dummy stubs for interrupt service routines
;

ext_int0:       reti

;
; External Interrupt 1 handler.
;

ext_int1:       reti


timer1_capt1:
timer1_comp1:
timer1_ovf1:
timer0_ovf:
uart_rx:
uart_dre:
uart_tx:
ana_comp:       reti


;
; MAIN BODY OF PROGRAM
;



;*************** main ***************
;
;
;
;************************************
;
;


start:  ldi     r16,RAMHIGH
        out     spl,r16         ;initialize stack to top of RAM

        rcall   initports_run   ;port b/d setup for RUN mode

        rcall   init_timer
        rcall   init_timer1     ;initialize timers

;
; check PD0 status for external input
;

st1:    ldi     r16,1           ;1/2 second wait
        rcall   waitR16seconds
        in      r16,pind        ;test EXT IN
        bst     r16,0
        brts    ext_input       ;if (PD0==1) use external input
```

```
;
; 19.2582 Hz strobe routine with 30-second timeout
;



main_prog:

        ldi     r16,1           ;initial value of PB0 output
        ldi     r17,1           ;value for toggling PB0 output


;
; *** Flash time limiter. Runs for up to 420 flashes (about 30 seconds)
;     before auto-shut off.
;

        ldi     r20,4
        ldi     r21,112         ;Max 30 seconds (420 flashes)


main_loop:
        eor     r16,r17         ;toggle PB0
        out     portb,r16       ;write the result

;
;Delay loops. f=14.025 Hz, T=0.071301247772 s,
; T/2 = 35,650.6238 us
;

        ldi     r19,166         ;do 165 loops
                                ;2+165(767+3) T
                                ;total for outer, inner loops

wait_outer:
        ldi     r18,$ff         ;inner loop = 255*3 + 2 T
wait:   dec     r18
        brne    wait

        dec     r19
        brne    wait_outer

;
; This code uses up the remaining 560T
;

        ldi     r18,40          ;187*3 + 2 T = 560 T
wait2:  dec     r18
        brne    wait2

;
; *** Timer for auto-shutoff here. ***
;
; Counts down from the assigned limit value, then shuts off the strobe when this is
; reached.
;

        dec     r21
        brne    main_loop
        dec     r20
        brne    main_loop
```

```
;
; 30 seconds have passed, shut it off
;
;

        clr     r16
        out     portb,r16

end:    rjmp    end                     ;that's all


;
; Ext input routine. NO time limit!
;

ext_input:
        in      r16,pind
        andi    r16,$01                 ;isolate PD0
        out     portb,r16
        rjmp    ext_input



;
; Wait for the number of half-seconds specified by R16.
;
;
WaitR16Seconds:

        ldi     r17,$60

WaitR16Seconds_Inner:
        ser     r24             ;about 3 mS delay
        rcall   wait_r24
        dec     r17
        brne    WaitR16Seconds_Inner

        dec     r16
        brne    WaitR16Seconds

        ret


;
;Wait for the number of Timer/Counter0 cycles specified by R24.
;R25 is altered.
;Timer/Counter0 must be initialized before calling.
;


wait_r24:
        clr     r25
        out     tcnt0,r25
wr24:   in      r25,tcnt0
        cp      r25,r24
        brlo    wr24
        ret
```

```
;*****************************************************
;
; EeWrite: Write to the EEPROM. R16 holds the data, R17 holds
; the address.
;
;*****************************************************

eewrite:       sbic    eecr,1          ;test bit 1 in eecr (EEWE)
               rjmp    eewrite         ;wait if not ready
               out     eedr,r16        ;setup the data
               out     eear,r17        ;setup the address
               sbi     eecr,2          ;set EEMWE (specific to '2313)
               sbi     eecr,1          ;set EEWE (pgm location)
               ret


;*****************************************************
;
; EeRead: Read from the EEPROM. R16 returns the data, R17 holds
; the address.
;
;*****************************************************

eeread:        sbic    eecr,1          ;test EEWE (should be 0 if ready to read)
               rjmp    eeread          ;wait if not ready to read
               out     eear,r17        ;output address
               sbi     eecr,0          ;turn on read-enable
               in      r16,eedr        ;get data
               ret



;
;******************** init_timer ******************
;
;Initialize the timer to be clocked directly from the T0 pin,
;but not actually generate any interrupt.
;
;R16 is destroyed.
;
;*************************************************
;

init_timer:

       ldi     r16,0b00000011 ;D7-D3=reserved=0

                              ;D2=CS02=0
                              ;D1=CS01=1 --> CLK/64
                              ;D0=CS00=1
                              ;
       out     tccr0,r16
       ret
```

```
;
;******************** init_timer1 ******************
;
;Initialize the timer1 to be clocked directly from the T0 pin,
;using the internal /1024 prescaler, but generate no interrupts.
;
;R16 is destroyed.
;
;***********************************************
;

init_timer1:

        clr     r16
        out     tccr1a,r16      ;d7=COM1A1=0
                                ;d6=COM1A0=0 --> Timer not connected to pin OC1
                                ;
                                ;d5-d2 RESERVED
                                ;
                                ;d1=PWM11=0
                                ;d0=PWM10=0  --> PWM operation is disabled

        ldi     r16,0b00000101

        out     tccr1b,r16      ;d7=ICNC1=0 --> Disable input noise canceller
                                ;d6=ICES1=0 --> ICP falling edge for capture mode
                                ;d5-4 RESERVED
                                ;d3=CTC1=0  --> No clear on compare match
                                ;
                                ;d2=CS12=1
                                ;d1=CS11=0
                                ;d0=CS10=1  --> /1024 prescaler selected

        ret


;
;******************** initports_run ******************
;
; Set the I/O ports for actively running I/O tasks.
;
; All pins are outputs, but only PB0 is actually used.
;
;***********************************************
;

initports_run:
        ldi     r16,0b11111111 ; PB0-7 are outputs
        out     ddrb,r16       ; Program port B DDR

        clr     r16
        out     portb,r16      ; force all port B pins low


        ldi     r16,0b11111110 ;PD0 = input, PD1-6 are outputs
        out     ddrd,r16       ;
        clr     r16
        out     portd,r16      ; force all port D pins low
                               ; and disable pullup on PD0
        ret
```