Watchdog

Richard Walker

February 23, 2009

Abstract

A hardware device for alerting operators to a firmware-lockup condition in Czochralski crystal growth furnace. By diagnosing lockup in a simple hardware add-on, the operator is able to reset the crystal growth algorithm and prevent disruption to the growth process.

0.1 Need for lockup detection

A Czochralski crystal growth furnace control system may lockup for various reasons. Several common conditions have been identified and corrected or improved. These include 1) ground loops, 2) unstable power supply voltages, 3) heat buildup in key communication ICs, 4) timeout conditions in printer subsystem, and 5) static sensitivity.

Several other factors are difficult to address due to the age and condition of the control system. These include 1) intermittent connection due to wear of plating in connectors, 2) oxidation and polarization of IC socket plating, 3) poorly thought-out grounding systems in a legacy system creating overvoltage conditions at sensitive nodes, and 4) possible bugs in the control system hardware and software (eg: asynchronous sampling of ADCs producing unpredictable control input).

Because it is not practical or possible to solve all the lock-up issues given the nature of a legacy control system, it is desirable to alert the operators to lockup when it occurs. In most cases, the system can simply be reset and the growth process will continue without incident. If, however, the operators do not identify and correct lockup within a critical period of time, the growth parameters will likely have drifted far enough to have created a defect in the crystal. This will require either remelting the crystal, or simply tailing off the growth to salvage a partial initial portion of the growth. In either case, an uncaught lockup event is costly.

0.2 Theory of operation

The basic idea for detecting lockup is to look for a "heartbeat" that is continuous under normal operation. A good candidate for this is the normal update process for the control CRT. Under normal run conditions, the control CRT is constantly updated with the process time, temperature, line currents, diameter, and so forth.

The watchdog sits in series between the CPU and the CRT emulator board. It is designed to simply snoop on any RS-232 transmission to the system monitor. The data rate is 9600 baud with no parity.

Everytime a character is sensed, a timer is reset. If the timer reaches 30 seconds without any character received, an audible alarm is sounded.

The alarm will continue to sound until a character is again sensed on the CRT input line. Reception of a character will immediately silence the watchdog. Under normal conditions, the watchdog will automatically detect lockup, send an alarm, and then reset back to idle condition with no explicit operator intervention beyond that required to address the lockup itself.

0.3 Implementation Details

To simplify operation, the watchdog has been designed to plug into two existing connectors. This method of installation requires no wiring or mechanical modifications. The connectors themselves provide all the mounting integrity required.

The watchdog board is installed on the BOS panel and snaps into both the existing CRT connector and the currently unused printer connector.

To save money and simplify the circuit design, a microchip P16F88 processor is used. This processor is approximately $5 and includes the required RS232 interface, a counter-timer subsystem, and is able to run off an internal 8MHz ring-oscillator to save parts count.

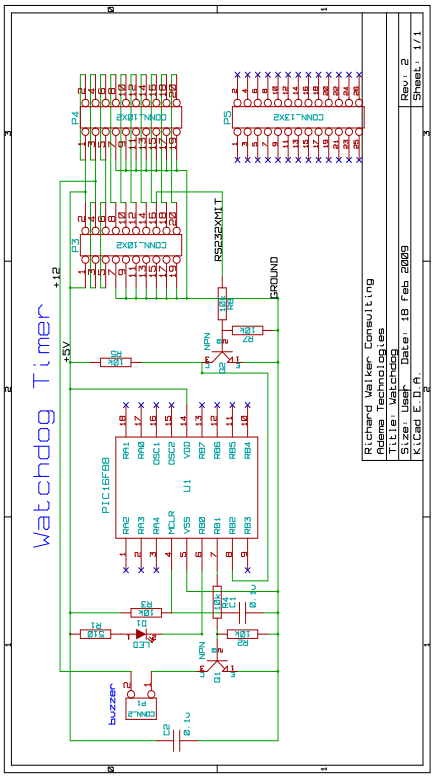
The RS232 signal is level-shifted to 5V TTL levels using an NPN transistor.

The Sonalert alarm module is run off 12V power for maximum volume. An NPN logic transistor is used to interface 12V driver to the 5V microprocessor output.

0.4 PCB layout

The circuit board was designed and layed out in Kicad .[1]

[1] www.kicad.sourceforge.net



0.5 Bill of Materials

Qty	part	description	Digikey	unit price
2	C1-2	0.1uf cap	399-4151-ND	$0.16
1	D1	LED	P605-ND	$0.38
1	P1	Buzzer	688-1029-ND	$4.51
1	P3	CONN 10x2M	MHB20K-ND	$2.00
1	P4	CONN 10x2F	MSPV20-ND	$3.37
2	Q1-2	NPN	2N3904FS-ND	$0.11
1	R1	510 ohm	CF1/4510JRCT-ND	$0.03
6	R2-8	10k ohm	CF1/410KJRCT-ND	$0.03
1	U1	PIC16F88	PIC16F88-I/P-ND	$5.00
1	S1	18DIP socket	AE9995-ND	$0.54

0.6 Firmware

The watchdog is programmed directly in Microchip assembly language. The resulting file is assembled into a HEX file using the GNU pic programming tool gpasm.[2]. The programming is done using the picp programming software[3] and the Olimex MPASM programmer PG-00003 [4].

```
;*****************************************************************
;       PIC16F874
;       1.      Have the processor generate an interrupt every 1 ms.
;       2.      Toggle Port A <4> every 500 ms.
;       3.      Keep a count of interrupts in timer0-3 for real-time
;
;    Notes:
;       This program generates an interrupt every 1 ms.
Upon interrupt
;       it updates a counter.
;       PORTA<4> is an output pin.  It is toggled every 500 ms.
;
;       The processor runs off its internal 8MHz ring oscillator
;
;       RS-232 input is RB2/SDO (pin 8)
;       RS-232 output is RB5 (pin 11)
;       Pushbutton is RB7
;*****************************************************************

        list      p=16f88             ; list directive to define processor
        #include <p16f88.inc>         ; processor specific variable definitions
        errorlevel      1,-(305)

;       __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _LVP_OF
        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _INTRC_IO & _LVP_

; The particular choice given above turns code protection off, watch dog
; timer off, brown-out reset disabled, power-up timer enabled, Internal RC
; oscillator with i/o pins selected, flash program memory write disabled,
; low-voltage in-circuit serial programming disabled, and data EE
; memory code protection off.

;***** VARIABLE DEFINITIONS
        cblock 0x20             ; define a series of variables.
                w_temp          ; isr context storage
                status_temp     ; isr context storage
```

[2] http://gputils.sourceforge.net/
[3] http://home.pacbell.net/theposts/picmicro/
[4] http://www.sparkfun.com

```
                count2          ; The MSB of a 3-byte counter
                count1          ; counting the number of Timer 2 interrupts
                count0          ; The LSB of the same 2-byte register.
                flag            ; set to one after timeout
                sportb          ; keep track of portb state
                timer0          ; clock timer
                timer1
                timer2
                timer3
        endc

; Bits within PORTA
ToggleBit       equ     4               ; An output bit, toggled every 500 ms.
PortAMask       equ     B'00000000'     ; We'll make all of PORT A an output.

;*****************************************************************

reset:          ORG     0x000           ; processor reset vector

                clrf    PCLATH          ; ensure page 0 is used
                goto    init            ; go to beginning of program

isr:            ORG     0x004           ; interrupt vector location

                movwf   w_temp          ; save current W register contents
                swapf   STATUS,W        ; move status register into W register
                clrf    STATUS          ; select Bank 0
                movwf   status_temp     ; save off contents of STATUS register
                                        ; There's no need to save PCLATH since
                                        ; we'll stick to Bank 0 of program memor

                                        ; isr code can go here or
                                        ; be located as a call subroutine elsewh

                btfss   PIR1,TMR2IF     ; If Timer 2 caused the interrupt, handl
                goto    endint
                call    Timer2

endint:         swapf   status_temp,w   ; retrieve copy of STATUS register
                movwf   STATUS          ; restore pre-isr STATUS register conten
                swapf   w_temp,f
                swapf   w_temp,w        ; restore pre-isr W register contents
                retfie                  ; return from interrupt

; ****************************************
;       Timer 2 Interrupt handler.
```

```
;
; decrements counter every ms until zero
; then sets beeper bit and count "flag"
; will then stall indefinitely until flag is cleared.
;

Timer2
        btfsc   flag,0          ; count down if beeper is off
        goto    endcnt

        decf    count0,F        ; Decrement the LSB of the 1-ms counter.
        skpz
        goto    endcnt
        decf    count1,F        ; ...
        skpz
        goto    endcnt

        bsf     PORTA,2         ; turn on beeper
        movlw   H'FF'
        movwf   flag

        clrf    count0          ; Reinitialize counters
        movlw   d'100'          ; d'16'=4 seconds
        movwf   count1
        clrf    count2          ; not used

endcnt
        BCF PIR1,TMR2IF ; Clear int flag and continue.
        return

; ********************************************************************
; START OF CODE to initialize the processor
; The initialization code goes here since we'll end up here shortly after a rese
; ********************************************************************

init    clrf    sportb          ; clear variables
        clrf    timer0
        clrf    timer1
        clrf    timer2
        clrf    timer3

        banksel OSCCON
        movlw   B'01110000'     ; set 8MHz internal oscillator
        movwf   OSCCON

        bcf     STATUS,RP0      ; Select Bank 0
```

```
        bcf     STATUS,RP1

        clrf    PORTA           ; Initialize Port A by clearing the output latch

        clrf    count0          ; Initialize counters
        movlw   d'100'
        movwf   count1
        clrf    count2

        clrf    flag            ; Turn off beeper flag

        banksel TRISA
        movlw   PortAMask       ; Initialize direction pins for Port A using TRI
        movwf   TRISA

        banksel ADCON0
        movlw   B'00000000'     ; ADC off
        movwf   ADCON0

        banksel ANSEL
        movlw   B'00000000'     ; All bits are digital
        movwf   ANSEL

; RB2/SDO (pin 8) RX input
; RB5 (pin 11) TX output
; RB7 pushbutton
; RB7,6,4 are interrupt sources

        banksel TRISB
        movlw   b'11110101'     ; RB0 INT input, RB2 input
        movwf   TRISB
;
; BAUD RATE SETTINGS
;
        banksel SPBRG
        ;movlw   d'129'         ; 9600 baud @ 20 MHz +0.16 err
        movlw   d'51'           ; 9600 baud @ 8 MHz int RC osc
        movwf   SPBRG

        banksel TXSTA
        movlw   b'00100100'     ; brgh = 1, SYNC=0
        movwf   TXSTA           ; enable async xmit, set brgh

        banksel RCSTA           ; bank 0
        movlw   b'10010000'     ; enable asyn rcvr, SPEN=1, CREN=1
        movwf   RCSTA
```

```
        banksel RCREG
        movf    RCREG,W
        movf    RCREG,w
        movf    RCREG,w         ; flush receive buffer

        banksel PORTA
        movlw   d'33'           ; send '!'
        call    putchar
        movlw   d'10'           ; send '\n'
        call    putchar

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        bcf     STATUS,RP0      ; Revert to Bank 0

; ***********************************************************************
; Initialize Timer 2
; Set up Timer 2 to generate interrupts every 1 ms.  Since we're assuming an ins
; cycle consumes 0.2 us (20MHz clock), we need an interrupt every 5000 instructi
; We'll set the prescaler to 4, the PR2 register to 125, and the postscaler to 1
This
; will generate interrupts every 4 x 125 x 10 = 5000 instruction cycles.
; ***********************************************************************

        CLRF    TMR2            ; Clear Timer2 register

        banksel INTCON          ; bank 0
        bsf     INTCON,PEIE     ; Enable peripheral interrupts

        banksel PIE1
        clrf    PIE1            ;
        bsf     PIE1,TMR2IE     ; enable timer 2 interrupts.

        banksel PIR1            ; bank 0
        CLRF    PIR1            ; Clear peripheral interrupts Flags
        movlw   B'01001001'     ; Set Postscale = 10, Prescale = 4, Timer 2 = of
        movwf   T2CON

        banksel PR2             ; bank1
        ;movlw   D'125'-1       ; Timer 2 to divide by125 (20MHz)
        movlw   D'50'-1         ; Timer 2 to divide by50 (8MHz)
        movwf   PR2

        banksel INTCON          ; bank0
        bsf     INTCON,GIE      ; Global interrupt enable.
```

```
        BSF     T2CON,TMR2ON    ; Timer2 starts to increment

; ********************************************************************************
; main()  This is the main program loop.

loop    call    getchar         ; stall waiting for a character
        clrf    flag            ; input clears the beeper
        bcf     PORTA,2         ; turn off beeper

        clrf    count0          ; Reinitialize counters
        movlw   d'100'          ; d'16'=4 seconds
        movwf   count1
        clrf    count2          ; not used

        banksel PORTA
        movlw   d'1'
        xorwf   PORTA,f

        goto    loop

; ********************************************************************************
; RS-232 output routine
; putchar - loops until char accepted
;-------------------------------------------------

putchar banksel PIR1
        btfss   PIR1,TXIF       ; xmit buffer empty?
        goto    putchar
        movwf   TXREG

        banksel PORTA
        return

;-------------------------------------------------
; RS-232 input routine
; getchar - loops until char received
; returns value in w
;-------------------------------------------------

getchar banksel PIR1
        btfss   PIR1,RCIF       ; got data?
        goto    getchar
        movf    RCREG,w         ; read data
        bcf     RCSTA,CREN      ; disable (clear any errors)
        bsf     RCSTA,CREN      ; re-enable
        return
```

```
END                                    ; directive 'end of program''
```