Pentium® III Processor Active Thermal Management Techniques

Application Note

August 2000

Order Number: 273405-001

Application Note

Contents

Figures

Tables

1.0 Overview

The maximum power consumption of Pentium® III processors can range from 12.2 W to more than 26 W. Embedded processors may easily reach their maximum specified die temperature during high load situations and when they are subjected to high ambient temperatures. For this reason, system designers should consider implementing an active thermal management solution to ensure that the processor die temperature is within specification.

This application note presents various methods of throttling the Pentium III processor to ensure that the processor remains within its temperature specification. This document also includes the following:

- Techniques on how to monitor and control the processor temperature in embedded systems.
- Instructions on how to access the various temperature sensors and power management features of a Pentium III processor chipset.
- A reference example of clock throttling using a microcontroller.

2.0 Measuring the Processor Temperature

A key part of active thermal management is to accurately measure the processor temperature. The temperature on a Pentium III processor can be accurately measured using the centrally located on-die thermal diode. The diode provides a method of measuring the temperature to ensure that the processor is within its tested specified operating temperature range. The anode and cathode of the on-die temperature diode can be interfaced to a thermal sensor component via two pins of the processor. The temperature of the diode can be computed from the following equation:

$$I_D = I_S(e^K - 1)\ ; \quad where \quad K = \frac{qV_D}{nT}$$

Where:

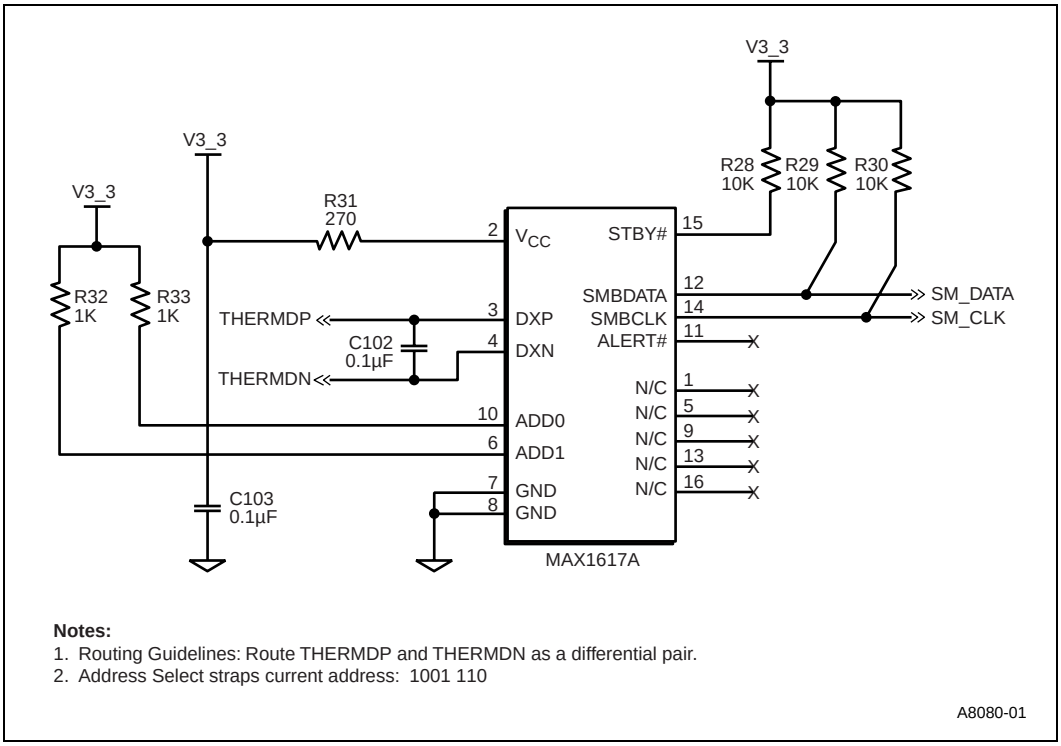
I_D = Current through diode
I_S = Reverse Saturation Current
V_D = Voltage drop across diode
n = Diode Ideality Factor (always >1) (1.0057 to 1.0125)
T = Temperature of diode
q = Constant

This equation can be simplified to the following:

$$T = \frac{qV_D}{n\ln\left(\frac{I_D}{I_S} + 1\right)}$$

Many thermal sensors are available that can be directly connected to the thermal diode on the processor, including the Maxim MAX1617A*, Analog Devices: ADM1021, ADM1022, ADM9240 or National Semiconductor LM84. An example using the MAX1617A to measure the temperature is illustrated in Figure 1.

Figure 1. Using the MAX1617A to Measure Temperature



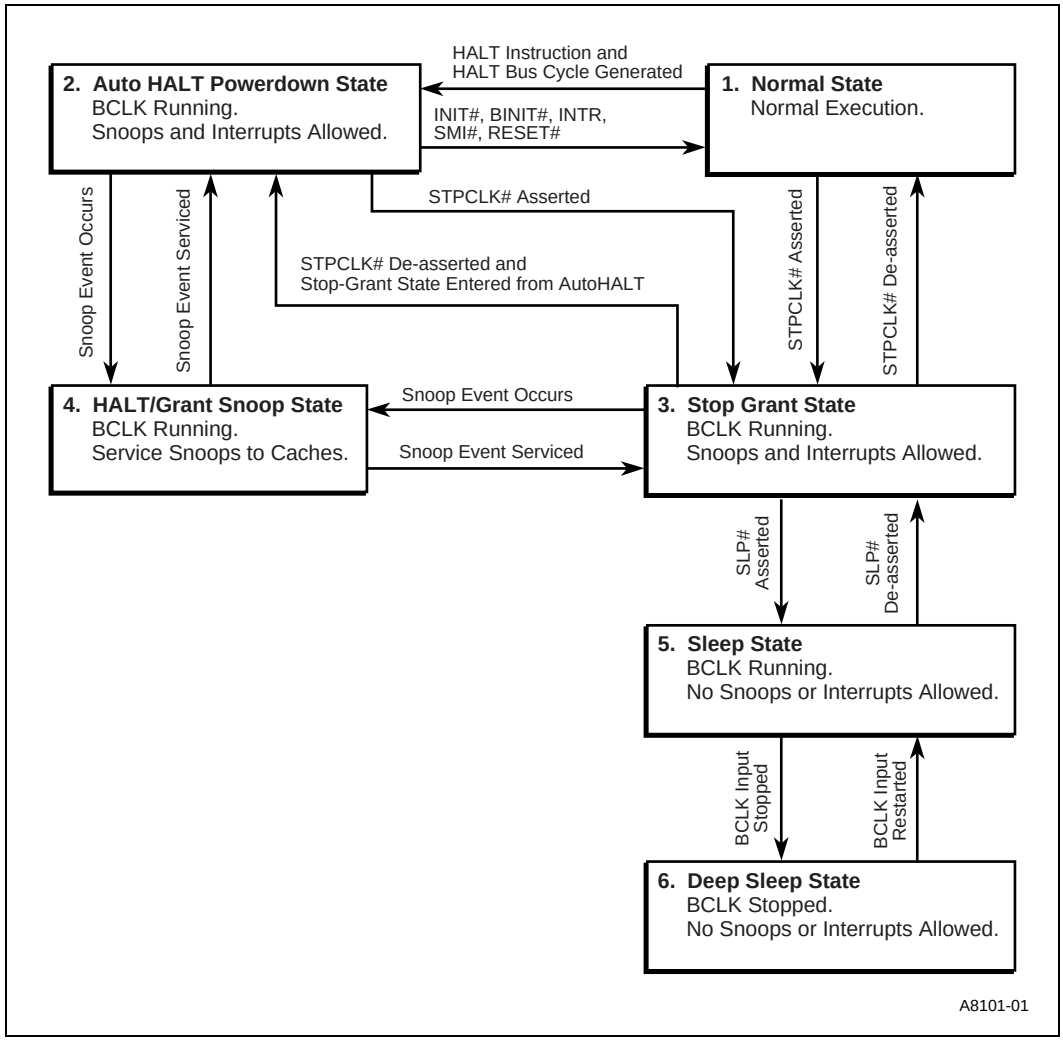
The thermal sensor can be used to report the die and ambient temperatures. These sensors can be interfaced to the Pentium III processor or a thermal microcontroller using various buses. The most common bus is the SMBus, which is a derivation of the I^2C* bus. The sensor can report the temperature at regular intervals over the SMBus, by polling from the Pentium III processor, or by generating an ALERT# if the temperature crosses a set trip point.

3.0 Lowering the Temperature of the Processor

Once the thermal sensor has detected that the processor temperature is out of the specified operating range, it is up to the microcontroller or chipset to begin throttling the processor into a lower power state. Another method to reduce the processor temperature is to have the microcontroller turn on a fan. Throttling the processor is accomplished by toggling the STPCLK# pin on the processor. When the STPCLK# pin is pulled low, the processor initiates a bus request and waits until it obtains ownership of the bus. After it obtains ownership of the bus, the processor places itself in the Stop Grant state which consumes less power (see Figure 2). In the Stop Grant state, clocks are stopped to the various logic blocks within the processor, except for the processor's PLL. To exit from the Stop Grant state, deassert STPCLK#. Only 10 internal bus clocks are required to exit.

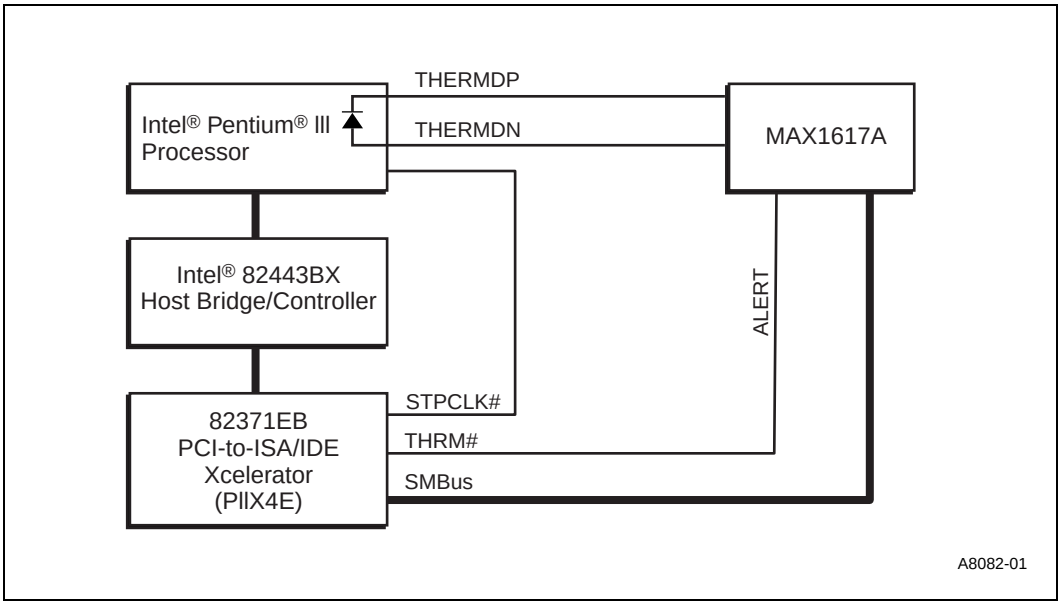
Note: To ensure that the Pentium III processor enters the Stop Grant state when STPCLK# is asserted, the processor must not be in QuickStart Mode. Please see the *Pentium® III Processor for the PGA370 Socket at 500 MHz to 1.0B GHz* datasheet (order number 245264) for more information on how to enter the Stop Grant state and on the limitations of QuickStart.

Figure 2. Stop Clock State Machine



The simplest implementation to control the temperature of the processor can be constructed using the temperature sensor and the PIIX4E southbridge to throttle the processor. The PIIX4E contains necessary logic to throttle the processor with a minimum amount of software. This example is illustrated in Figure 3.

Figure 3. Using the PIIX4E for Throttling

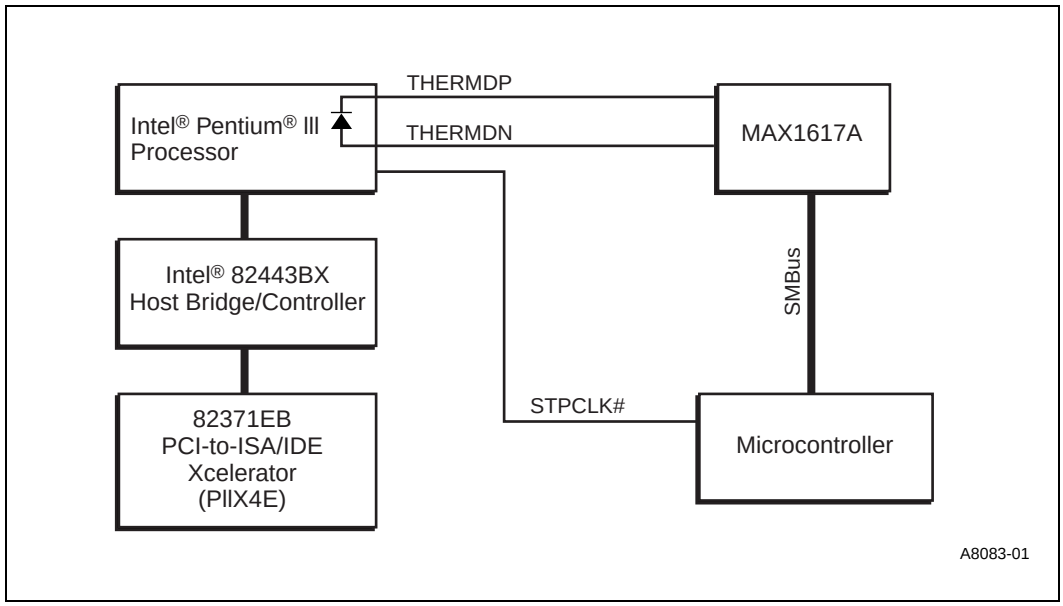


In this implementation, the processor programs the MAX1617A to generate the ALERT# signal (connected to THRM# on the PIIX4E) when the temperature approaches a critical zone. When the THRM# pin on the PIIX4E is held low for more than two seconds, the PIIX4E begins throttling the STPCLK# pin with the programmed duty cycle and with a period of 244 µS. To enable throttling, the THT_EN bit of the PCNTRL register must be set. The duty cycle can be programmed in 12.5% increments in the THRL_DTY field of the Processor Control Register (PCNTRL BASE+(10-13H)). To have the THRM# signal generate an SMI or SCI event, the THRM_EN bit of the GPEN register must be set. Refer to the *82371AB PCI-to-ISA/IDE Xcelerator (PIIX4)* datasheet (order number 290562) for more detailed information.

However, if the duty cycle of STPCLK# is not sufficient to cool down the processor, the operating system or System Management software must reprogram the duty cycle to slow down the processor. This may involve inserting interrupt routines.

Another method to manage the temperature of the processor uses an 8-bit microcontroller. In this implementation, the microcontroller polls the temperature sensor and generates an appropriate duty cycle for STPCLK#. The duty cycle on STPCLK# can decrease as the temperature increases. This ensures that the least amount of performance is sacrificed as the temperature increases. This implementation is illustrated in Figure 4. An advantage of this method is that it can be accomplished completely in hardware, without software interventions on the Pentium III processor or chipset.

Figure 4. Using an 8-bit Microcontroller to Manage Temperature



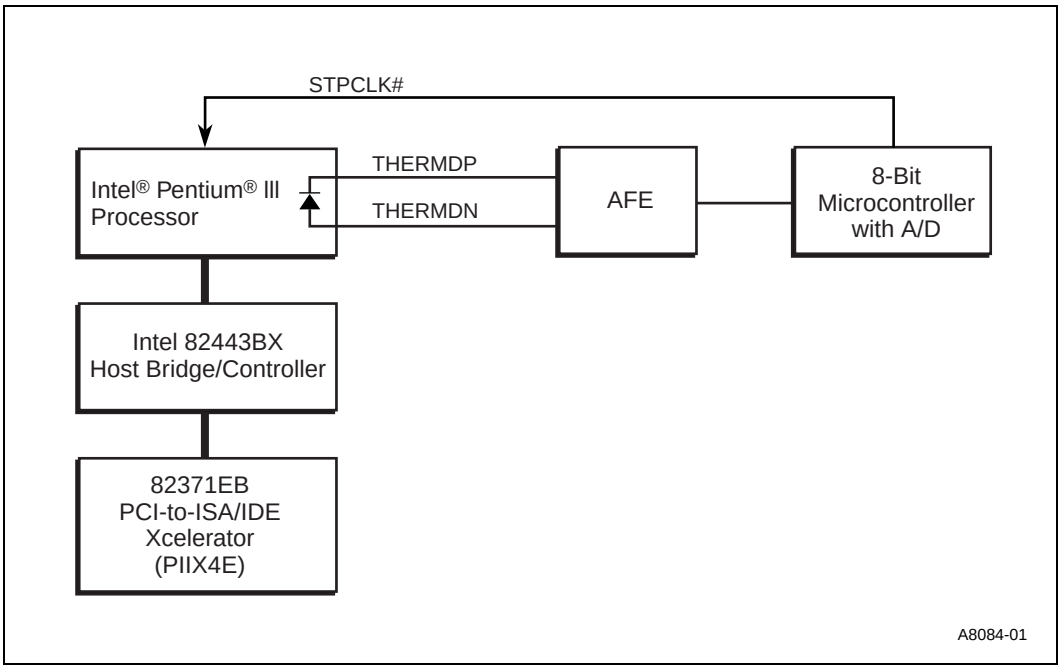
3.1 Clock Throttling Implementation with Microcontroller

A reference example of clock throttling is provided in this application note. The assembly code is included in Appendix A. There are several ways of implementing clock throttling using a microcontroller. One way is to use a temperature sensor such as the one provided by Maxim—the MAX1617A. This sensor uses the SMBus to communicate the temperature information and to receive setup information.

An 8-bit microcontroller interfaces easily using the SMBus, and based upon the temperature information, performs the clock throttling function. This is the method detailed in this application note (see Figure 4).

Another possible method reduces cost by removing the temperature sensor. An analog front-end is needed to provide a constant current source to the thermal diode and amplify to the signal. In this case, the thermal diode can be linearized in software or with a filter. This alternative method is shown in Figure 5.

Figure 5. Alternative Method Using an 8-bit Microcontroller with Integrated A/D and Analog Front-end



3.2 Reference Example

To demonstrate clock throttling with an Intel 700 MHz Pentium III processor, the following tools/components were used:

- Microchip PIC16F876 8-bit microcontroller (A/D, USART, 28 pins, I^2C)
- Intel® 440BX Scalable Performance Board Development Kit
- Microchip PICDEM-2 demoboard, a microcontroller development platform
- Microchip MPLAB-IDE, integrated development environment

The assembly language source code for the microcontroller is located in Appendix A.

3.3 Reference Source Code

The source code provided in this application note is not fully tested or guaranteed. The customer using this reference must test and validate the circuit and assembly code. It is provided strictly as reference or template code. It has been developed as a proof-of-concept of active thermal management. The logic flow of the code is explained in the following paragraphs.

The human interface to this application is a display terminal. The PICDEM-2 demo board has the hardware for an RS232 port. This peripheral is used by the microcontroller to output temperature and other information to a display terminal. A terminal program running on a PC can be used to display the information. In this way, the designer can take temperature readings using a thermocouple and compare it to the temperature data output by the microcontroller.

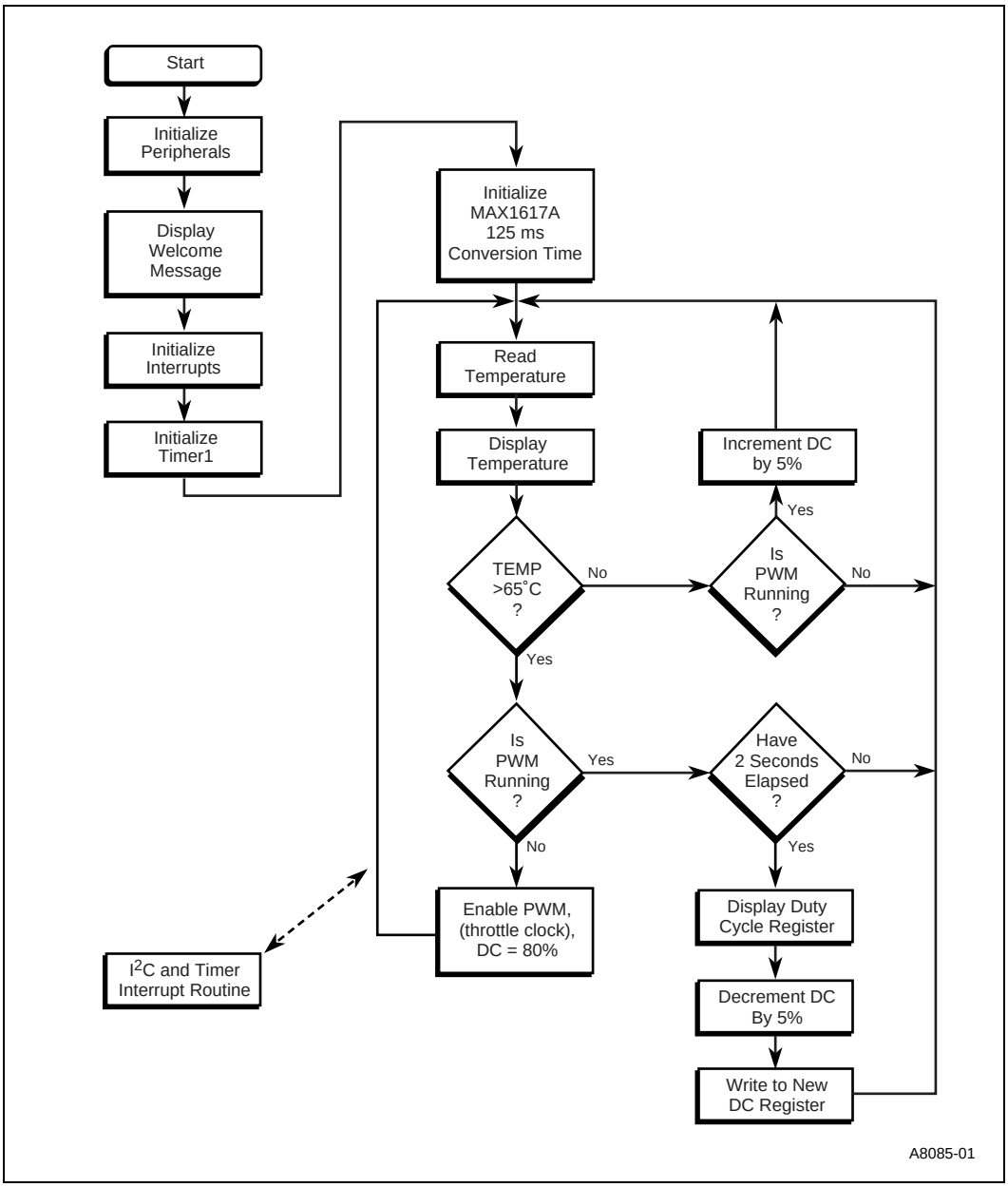
There is code that covers additional commands for the MAX1617A, which was not used in this application, but is provided in Appendix A for reference.

The flowchart in Figure 6 shows the main loop and interrupt service routine. The MAX1617A reads the thermal diode on the Pentium III processor every 0.125 seconds. Essentially, the microcontroller polls the MAX1617A thermal sensor every 0.25 seconds. The microcontroller outputs the temperature reading to the USART as ASCII text. When the junction temperature rises above TEMP_HI (in this case 65° C), the clock is throttled. The duty cycle register value (0-255) is output to the display via the RS232 interface on the demoboard (see mastri2c.asm in Appendix A.)

Throttling the clock involves generating a pulse width modulated (PWM) signal on the STPCLK# input of the Pentium III processor. The microcontroller can produce either an 8-bit PWM or a 10-bit PWM. For this application, only 8-bit resolution is needed. The values for the duty cycle register may be between 0 and 255. The duty cycle is initially chosen to be 80%. This means that the CPU effectively is running at 80% of its frequency.

Another timer on the microcontroller is used to keep track of how long the PWM is running. In this example, if two seconds have passed and the junction temperature is still above 65° C (TEMP_HI), the duty cycle is decreased by 5%. This continues until the temperature decreases below the threshold or the duty cycle is 0% (shutting off the clock completely). When the temperature decreases below the threshold of TEMP_HI, the duty cycle is increased at 5% increments. The temperature is read again, and the PWM duty cycle is adjusted either up or down.

Figure 6. Flowchart of Clock Throttling Software Program



3.4 Test Results

In order to test the effectiveness of the clock throttling solution and the performance impact to the Pentium III processor, the Intel 440BX Scalable Performance Board Development Kit was used. Figure 7 through Figure 14 show both performance vs. time and $T_{Junction}$ (junction temperature) vs. time. There are four sets of charts. Each set shows a different load on the processor. Windows* 98 was running in all cases. A 700 MHz Pentium III processor in the FCPGA package was used for all testing.

3.4.1 Case 1: Fan and Heatsink Installed, Windows* 98 Running

Figure 7 and Figure 8 demonstrate the initial condition (smallest load). Only Windows 98 is running and the heat sink and fan are both installed. Figure 7 shows performance versus time. Performance is depicted as a percentage of the maximum frequency of 700 MHz. For example, if the duty cycle is 75%, then the performance is 75% (an effective frequency of 525 MHz). Performance is calculated as follows:

$$performance = 100\% \bullet \frac{duty\ cycle\ register\ value}{2^8-1}$$

Time is shown in seconds. The junction temperature $T_{Junction}$, is read and displayed every 0.25 seconds. Notice the performance in this case is a maximum of 100%, which means there is no clock throttling.

In Figure 8, the junction temperature, $T_{Junction}$, is shown versus time. Notice the junction temperature is at a steady state of approximately 43° C. There is a constant load on the processor, therefore the junction temperature is constant.

The maximum junction temperature, $T_{Junction,}$ for a 700 MHz Pentium III processor is 80° C. As a note, the $T_{JUNCTIONOFFSET}$ is the worst-case difference between the thermal reading from the on-die thermal diode and the hottest location on the processor's core. For the Pentium III processor running at 700 MHz, this value is 4.1° C. This number should be added to the diode temperature reading before determining if the temperature is past the pre-determined threshold.

For this application, the junction offset was not added to the thermal diode temperature reading for simplicity. In any application of clock throttling, the junction offset must be considered in choosing the threshold temperature. In this application, the arbitrary threshold of 65° C was chosen, this gives plenty of guardband to compensate for the junction offset temperature.

Figure 7. Performance vs. Time, 700 MHz Pentium® III Processor, FCPGA, Fan On, No Thermal Stress Software

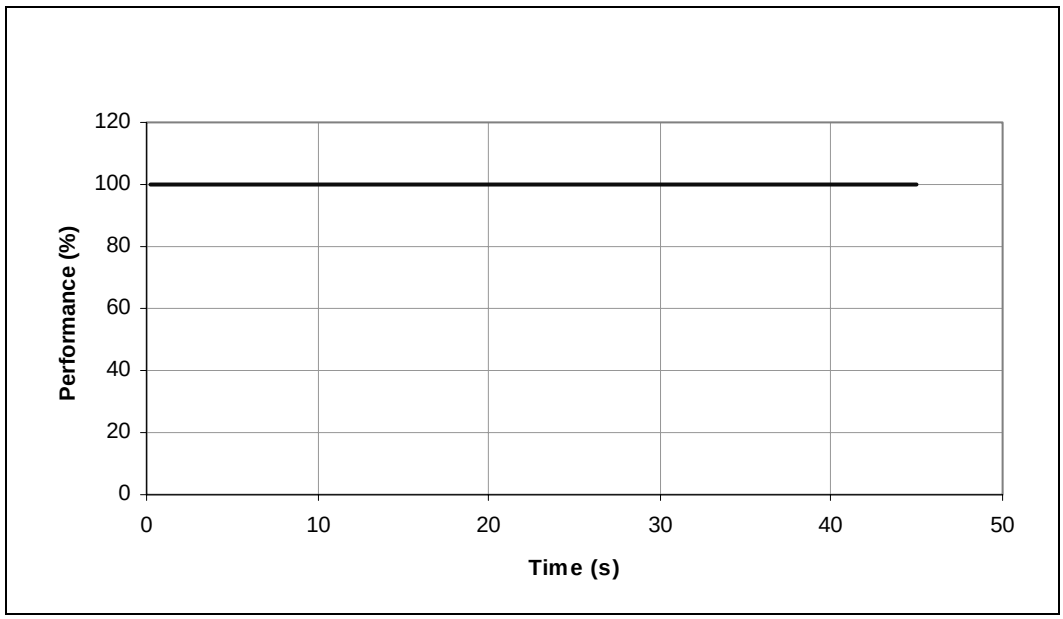
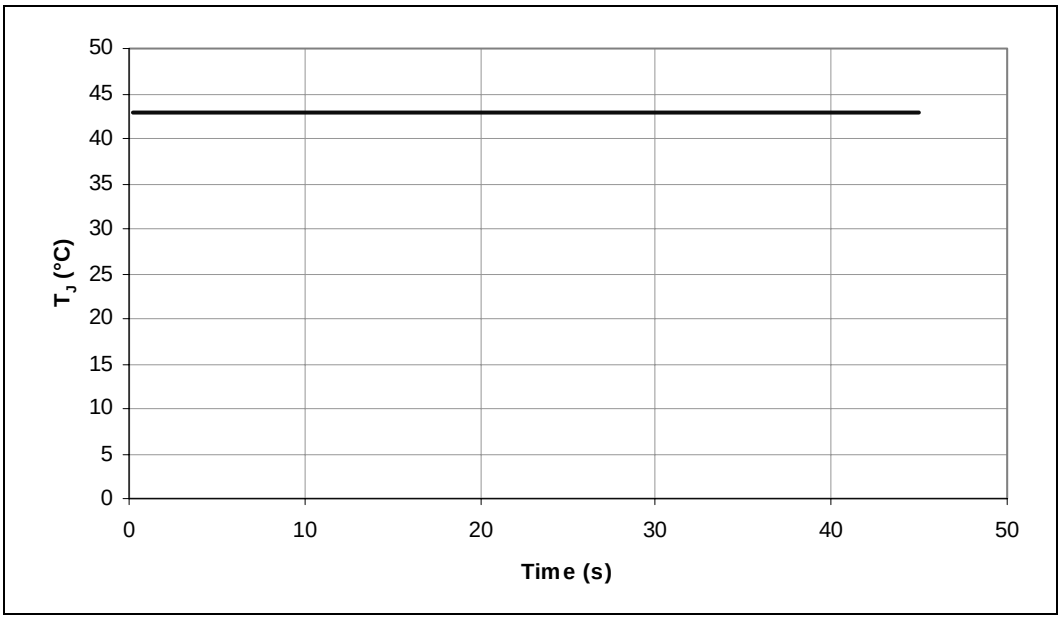


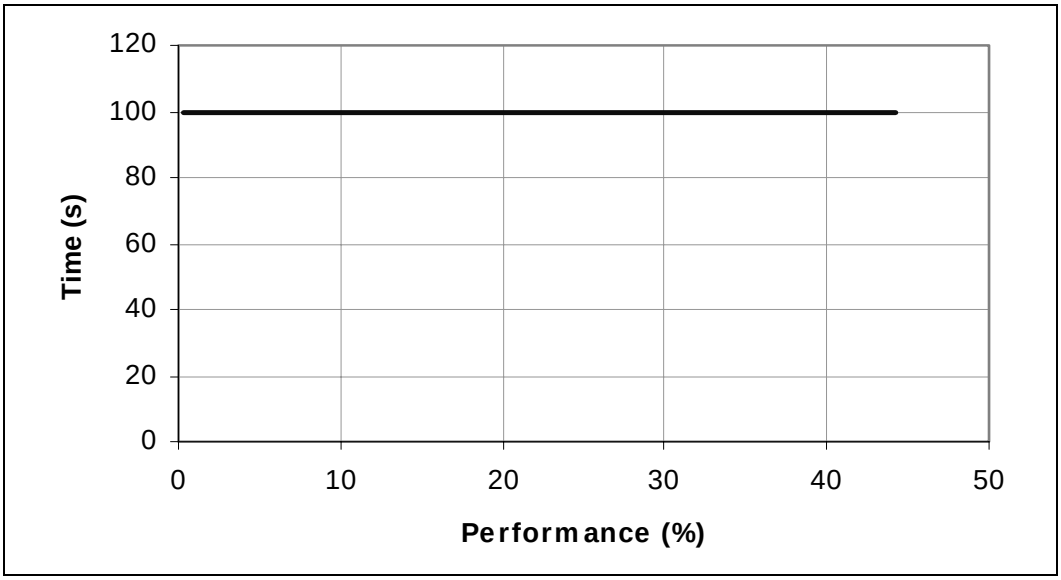
Figure 8. $T_{Junction}$ vs. Time, 700 MHz Pentium® III Processor, Fan On, No Thermal Stress Software



3.4.2 Case 2: Fan and Heatsink Installed, Windows 98 and Thermal Stress Software Running

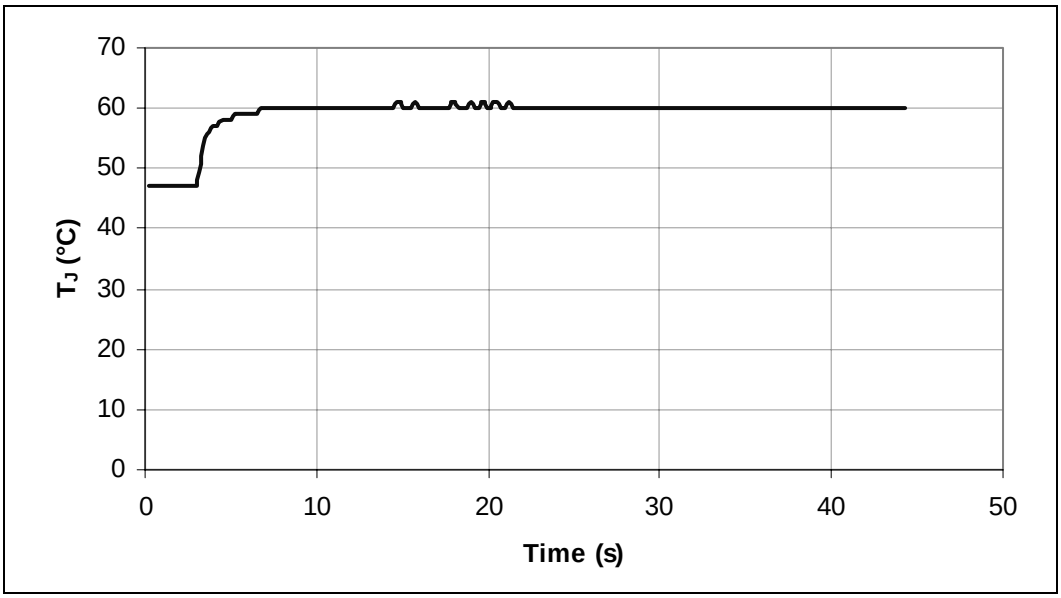
The next case shows Windows 98 and thermal stress software[1] running. Thermal stress software is an application that maximizes the load on the CPU. In Figure 9, the performance is maintained at 100%. Figure 10 shows when thermal stress software is first executed at approximately t=3 seconds, the temperature increases to 60° C at approximately t=8 seconds. The temperature remains in steady state at 60° C. Since the threshold of 65° C is not reached, the clock is not throttled and maximum performance is maintained.

Figure 9. Performance vs. Time, 700 MHz Pentium® III Processor, FCPGA, Fan On, Thermal Stress Software Running



1. Please contact your Intel Field Sales office for more information.

Figure 10. $T_{Junction}$ vs. Time, 700 MHz Pentium® III Processor, FCPGA, Fan On, Thermal Stress Software Running



3.4.3 Case 3: Fan and Heatsink Removed, Windows* 98 Running

Figure 11 and Figure 12 show what happens when the fan and heatsink are removed from the processor. The processor is initially running at 700 MHz at t=0 seconds. The fan is removed at approximately t=25 seconds (Figure 11). Here the performance drops by 5%, then it continues to drop at 5% increments until about t=48 seconds (Figure 11). At this point, the performance oscillates around 25%. Therefore, the processor is effectively running at 175 MHz with no fan or heatsink. Figure 12 shows when the fan is removed the temperature increases rapidly from t=25 seconds to t=33 seconds. Here it reaches its maximum temperature of 87° C, which is beyond the maximum allowable spec of 80° C. Notice the correlation between Figure 11 and Figure 12, as the temperature increases the duty cycle (performance) decreases. The temperature then decreases slowly at around t=35 seconds. It continues to decrease along with the duty cycle until it crosses the threshold temperature of 65° C. At t=48 seconds, the steady state temperature of 65° C and performance of 25% are maintained.

Figure 11. Performance vs. Time, No Fan, No Heatsink, 700 MHz Pentium® III Processor, FCPGA, No Thermal Stress Software

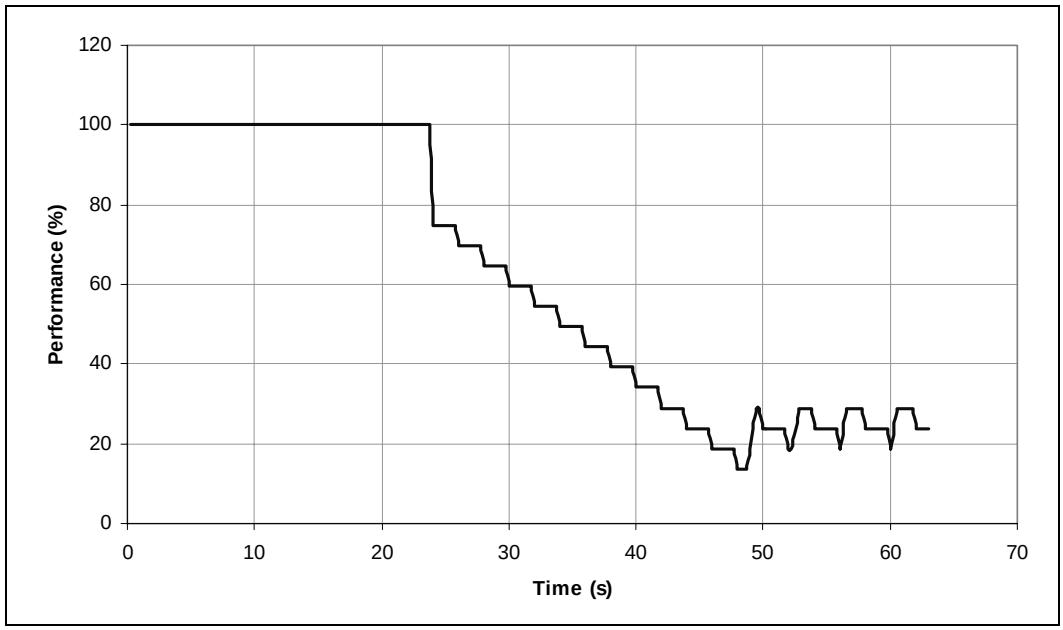
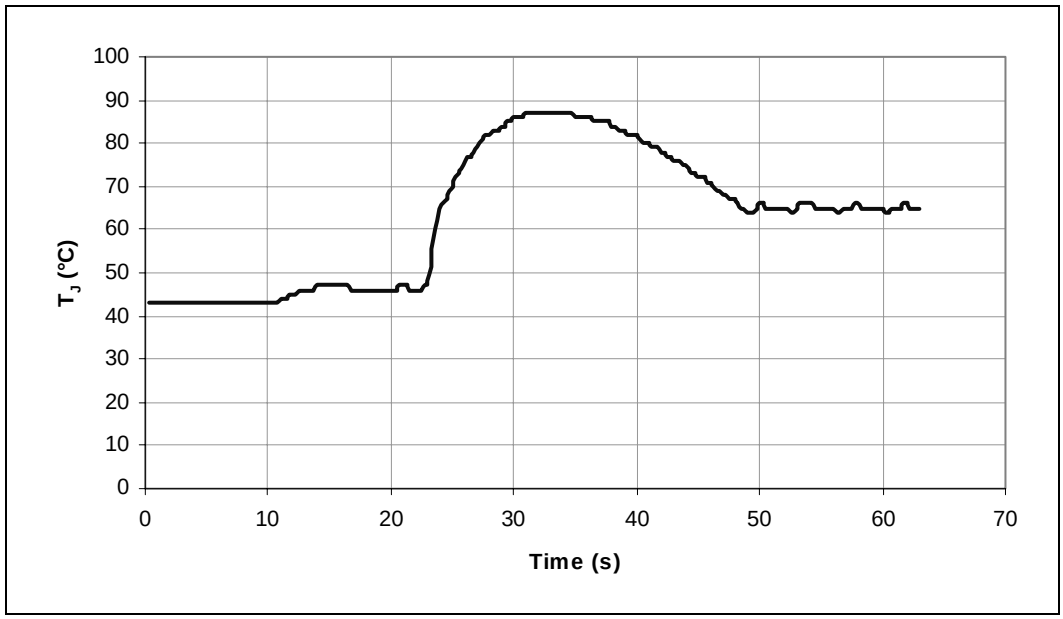


Figure 12. $T_{Junction}$ vs. Time, 700 MHz Pentium® III Processor, FCPGA, No Fan, No Heatsink, No Thermal Stress Software



3.4.4 Case 4: Fan and Heatsink Removed, Windows* 98 and Thermal Stress Software Running, Worst Case

Figure 13 and Figure 14 show the worst case when the fan and heatsink are removed and thermal stress software is running. First the fan and heatsink are removed at around t=3 seconds. Some time later thermal stress software is executed. Figure 13 shows the performance steadily decreasing until a steady state temperature is reached at 65° C. This occurs at around 40 seconds. Notice in Figure 13 that the performance is worse than the previous case. Here the performance is oscillating around 15%-20%, which yields an effective frequency of 105 MHz - 140 MHz. The processor is protected from overheating (thermal trip) by rapidly reducing the clock frequency. An alert signal could be generated, lighting an LED, for fan replacement.

Figure 13. Performance vs. Time, 700 MHz Pentium® III Processor, FCPGA, No Fan, No Heatsink, Thermal Stress Software Running

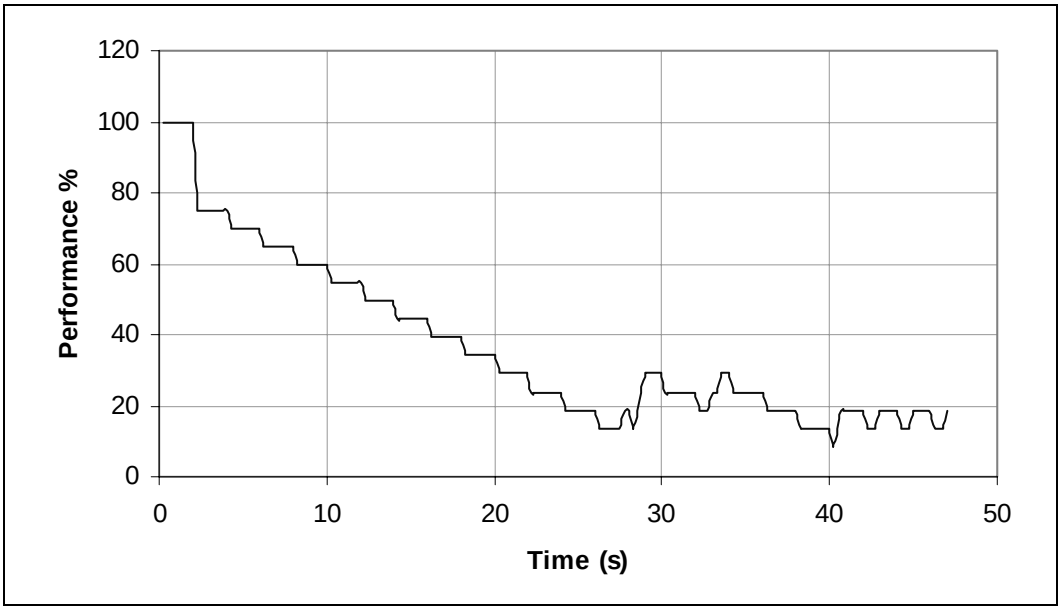
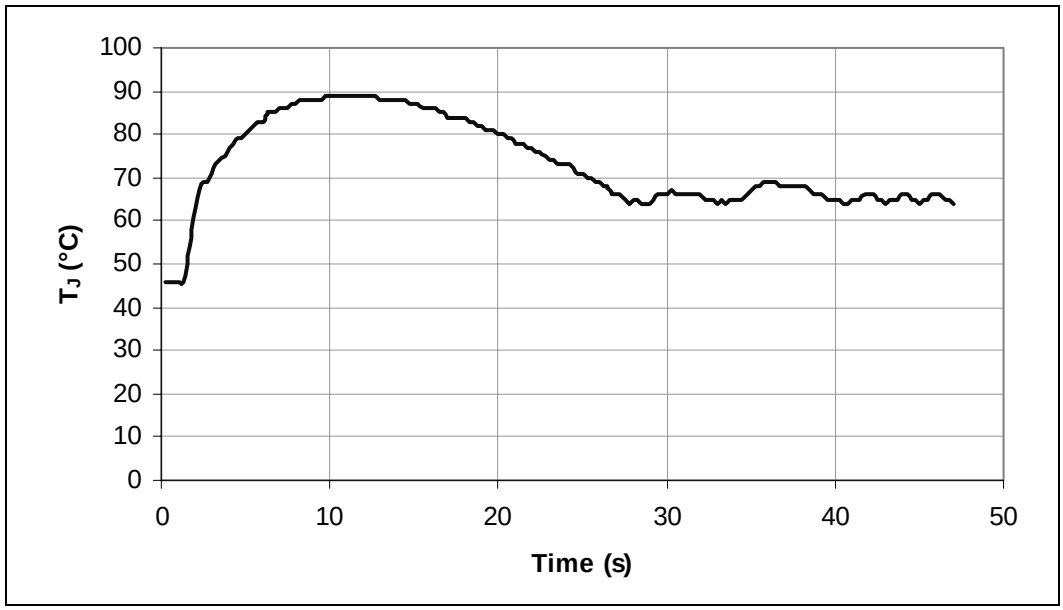


Figure 14. $T_{Junction}$ vs. Time, 700 MHz Pentium® III Processor, FCPGA, No Fan, No Heatsink, Thermal Stress Software Running



4.0 Customizing the Source Code

The source code can be modified for your application. For example, to change the threshold temperature, one constant would need to be changed: TEMP_HI. A duty cycle decrement/ increment value of 5% was chosen (which is 13 decimal in an 8-bit range). Also, the initial duty cycle was arbitrarily chosen to be 80% (204 decimal). These values can be changed by editing the assembly source code and updating the constants listed in Table 1. The table lists constants that control the clock throttling routine:

Table 1. Clock Throttle Constants

Constant	Init	Purpose
DC_DEC_VALUE	13	Duty cycle decrement value. Decrementing the duty cycle by 13, in an 8-bit PWM, will yield a 5% change in duty cycle.
DC_INIT	204	Duty cycle initialization value yields approximately 80% duty cycle with 8-bit PWM.
TEMP_HI	65	Arbitrary threshold of 65° C.

There are several files that make up the clock throttling program. Table 2 describes each file.

Table 2. File Types and Purpose for Clock Throttling Program

Filename	Type	Purpose
mastri2c.asm	assembly	Main file that contains the control loop for the clock throttling routine.
i2ccomm.asm	assembly	Contains I^2C master routines - see comments in file for subroutines.
init.asm	assembly	Initializes ports.
mastri2c.inc	include	Main routine register type declarations.
flags.inc	include	Flag bit declarations for I^2C status, PWM status, etc.
i2ccomm.inc	include	Defines variables used in I^2C routines.
i^2ccomm1.inc	include	Defines the I^2C function and variable types and scope for linker.
max1617a.inc	include	Defines command constants for MAX1617A.
p16f876.inc	include	Defines registers, etc. for microcontroller.
16f873.lkr	linker	Linker script file.

For more information, see the following references:

82371AB PCI-to ISA/IDE Xcelerator (PIIIX4) datasheet (order number 290562)

Intel 82371EB (PIIIX4E) Specification Update (order number 290635)

SMBus 1.1 Specifications can be found at the following Web site:
http://www.sbs-forum.org/smbus/specs/

For more information on the MAX1617A temperature sensor, see the following Web site:
http://www.maxim-ic.com/

5.0 Summary

Active thermal management techniques using an embedded microcontroller are very effective and easily implemented. The benefit of throttling the clock with a strictly hardware solution is that the solution is operating system software independent. With some analysis and experimentation, a more efficient algorithm could be used to throttle the clock. These solutions should be tailored to the specific application. This application note provides a template and example to get started with developing a custom clock throttling hardware solution.

Appendix A Software Listings

```
;*****************************************************************
;   Filename:       mastri2c.asm                                  *
;   Date:           04/04/2000                                    *
;   Revision:       005                                           *
; -004                                                            *
;  -adds duty cycle variable PWM_DC.  This DC register           *
;   will be decreased if the temperature keeps rising past 65 deg.*
;  -If temp falls below threshold then Duty Cycle = 100%          *
;                                                                 *
; -005 Increases Duty Cycle by 5% if temp falls below threshold   *
;                                                                 *
;                                                                 *
;   Tools:          MPLAB   5.00.00                               *
;                   MPLINK  2.00.11                               *
;                   MPASM   2.40.00                               *
;                                                                 *
;                                                                 *
;*****************************************************************
;                                                                 *
;   System files required:                                        *
;                                                                 *
;                   mastri2c.asm                                  *
;                   i2ccomm.asm                                   *
;                   init.asm                                      *
;                                                                 *
;                   mastri2c.inc                                  *
;                   i2ccomm.inc                                   *
;                   i2ccomm1.inc                                  *
;                   flags.inc                                     *
;                                                                 *
;                   p16f876.inc                                   *
;                   16f873.lkr     (modified for interrupts)      *
;                                                                 *
;*****************************************************************  *
;                                                                 *
;   Notes:                                                        *
;                                                                 *
;                                                                 *
;   Device Fosc -> 4.00MHz                                        *
;WDT -> off                                                       *
;Brownout -> on                                                   *
;Powerup timer -> on                                              *
;Code Protect -> off                                              *
;                                                                 *
;   Interrupt sources -                                           *
;                       1. I2C events (valid events)              *
;                       2. I2C Bus Collision                      *
;                                                                 *
;                                                                 *
;                                                                 *
;*****************************************************************

list      p=16f876                    ; list directive to define processor
#include <p16f876.inc>                ;processor specific variable definitions
__CONFIG (_CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_ON &
_LVP_OFF & _CPD_OFF)

#include  “mastri2c.inc”             ;
#include  “i2ccomm1.inc”             ; required include file
        #include   “max1617a.inc”
error level -302
```

```
#define  ADDRESS     0x4e              ;Slave I2C address
#define  TEMP_HI     0x41              ; Temperature High threshold is 65 deg C

#define  INIT_DC     100               ; Initial duty cycle is 100%, full speed

DC_DEC_VALUE equ     0x0D              ; 8 bit PWM, duty cycle decrement value 5% of256


DC_INIT     equ     0xCC               ; initialize duty to 80%



;-----------------------------------------------------------------------
;   ******************** RESET VECTOR LOCATION  ********************
;-----------------------------------------------------------------------
RESET_VECTOR  CODE    0x000             ; processor reset vector
              movlw  high  start        ; load upper byte of 'start' label
              movwf  PCLATH             ; initialize PCLATH
              goto   start              ; go to beginning of program


;-----------------------------------------------------------------------
;  ******************* INTERRUPT VECTOR LOCATION  *******************
;-----------------------------------------------------------------------
INT_VECTOR   CODE    0x004             ; interrupt vector location
              movwf   w_temp           ; save off current W register contents
              movf    STATUS,w         ; move status register into W register
              clrf    STATUS           ; ensure file register bank set to 0
              movwf   status_temp      ; save off contents of STATUS register
              movf    PCLATH,w
              movwf   pclath_temp      ; save off current copy of PCLATH
              clrf    PCLATH           ; reset PCLATH to page 0


; TEST FOR COMPLETION OF VALID I2C EVENT
              banksel  PIE1            ; select SFR bank
              btfss    PIE1,SSPIE      ; test if interrupt is enabled
              goto     test_buscoll
              banksel  PIR1            ; select SFR bank
              btfss    PIR1,SSPIF      ; test for SSP H/W flag
              goto     test_buscoll    ; no,
              bcf      PIR1,SSPIF      ; clear SSP H/W flag
              call     service_i2c


; TEST FOR I2C BUS COLLISION EVENT
test_buscoll
              banksel  PIE2            ; select SFR bank
              btfss    PIE2,BCLIE      ; test if interrupt is enabled
              goto     timer1test
              banksel  PIR2            ; select SFR bank
              btfss    PIR2,BCLIF      ; test if Bus Collision occured
              goto     timer1test
              bcf    PIR2,BCLIF        ; clear Bus Collision H/W flag
              call   service_buscoll   ; service bus collision error

timer1test
              banksel  PIE1
              btfss    PIE1,TMR1IE
              goto     exit_isr
              banksel  PIR1
              btfss    PIR1,TMR1IF
              goto     exit_isr
              bcf      PIR1,TMR1IF     ;timer1 rolled over

              banksel  sflag_event
              bsf      sflag_event,tmr1_ovr ;set the timer1 overflow flag
```

```
                banksel  timer1count
                incf     timer1count,f

                movlw    0x02
                banksel  PORTB
                xorwf    PORTB

exit_isr
                clrf    STATUS                  ; ensure file register bank set to 0
                movf    pclath_temp,w
                movwf   PCLATH                  ; restore PCLATH
                movf     status_temp,w          ; retrieve copy of STATUS register
                movwf   STATUS                  ; restore pre-isr STATUS register
contents
                swapf    w_temp,f               ;
                swapf    w_temp,w               ; restore pre-isr W register contents
                retfie                          ; return from interrupt



;--------------------------------------------------------------------------
;   ****************** MAIN CODE START LOCATION  ******************
;--------------------------------------------------------------------------


MAIN    CODE



;*************************************************************************
;
;*  Convert Twos complement # to four ascii chars including sign
;* INPUT: twos_num = twos complement byte number
;* OUTPUT: sign_num = + or - ascii sign
;*         hundreds_num = the hundreds place in ASCII
;*         tens_num     = the tens place in ASCII
;*         ones_num     = the ones place in ASCII
;*         twos_num,w,w_temp,count = destroyed
;
;*************************************************************************
convert_twos
                btfss    twos_num,7             ; contains 8-bit twos complement
                goto     sign_positive
                movlw    '-'
                movwf    sign_num               ; store neg sign in ascii

                comf     twos_num,w             ; complement twos #
                 addlw   1                      ; add 1, to convert to integer
                movwf    twos_num               ; save integer
                goto     get_digits
sign_positive   movlw    '+'
                movwf    sign_num
get_digits

                movlw    d'100'
                movwf    w_temp
                call     get_digit
                movwf    hundreds_num
                movlw    d'10'
                movwf    w_temp
                call     get_digit
                movwf    tens_num
                movlw    0x30
                addwf    twos_num,w
```

```
                movwf    ones_num
                return

get_digit
                clrf     count
get_digit_loop  subwf    twos_num,w
                btfss    STATUS,C                ; when C = 1 result is postive
                goto     get_digit_exit
                incf     count,f                 ; result is positve, increment place
                movwf    twos_num
                movf     w_temp,w
                goto     get_digit_loop
get_digit_exit
                movf     count,w
                addlw    0x30                    ; convert num 0-9 to ascii

                return



;**** Initialize the USART *******
init_USART
                movlw    d'103'                 ; 2400 baud @ 4MHz, brgh=1
                banksel  SPBRG
                movwf    SPBRG
                bsf      TXSTA,BRGH
                banksel  RCSTA
                bsf      RCSTA,SPEN             ;enable the serial port
                return

;*********************************************************
;* Start
;*********************************************************
start
                call     init_ports              ; initialize Ports
                call     init_i2c                ; initialize I2C module
                call     init_vars               ; initialize variables
                call     init_USART

;************
                banksel  Str_High
                movlw    high Welcome_Msg
                movwf    Str_High
                movlw    low  Welcome_Msg
                movwf    Str_Low
                call     Puts

                call     Init_Interrupt_Masks
                movlw   b'00101001'              ;1:4 prescale, osc enabled,
                                                 ; synch, intern, enable timer1
                banksel  T1CON
                movwf    T1CON

                movlw    RCRA                    ; read conversion rate byte
                banksel  Max1617_Cmd
                movwf    Max1617_Cmd
                call     Send_Command_Max1617
                call     Display_Max1617_Results ; print out conversion rate

                movlw    WCRW                    ; write conversion rate byte
                banksel  Max1617_Cmd
                movwf    Max1617_Cmd
                movlw    0x07                    ; 125ms sample period
                banksel  Max1617_Data
                movwf    Max1617_Data
                banksel  i2c_rw
                bsf      i2c_rw,rw               ; write data, rw = 1
```

```
                bcf      i2c_rw,stop
                call Send_Command_Max1617

                movlw      RCRA                      ; Read conversion rate byte
                banksel    Max1617_Cmd
                movwf      Max1617_Cmd
                call       Send_Command_Max1617
                call       Display_Max1617_Results ; print conversion rate byte

                banksel    timer1count
                clrf       timer1count

Send_Temp
                movlw      RRTE                      ; read remote temperature
                banksel    Max1617_Cmd
                movwf      Max1617_Cmd               ; Save which command we used
                call       Send_Command_Max1617
                call       Display_Max1617_Results


                banksel    read_string
                movlw      TEMP_HI                   ; Is temp > TEMP_HI deg C ?
                subwf      read_string,w
                btfsc      STATUS,C                  ; If C=0, result is negetive,
temp< TEMP_HI threshold
                goto       Throttle
                banksel    sflag_event
                btfss      sflag_event,pwm           ; check if pwm running
                goto       Send_Temp                 ; if not then continue

                movlw      DC_DEC_VALUE              ; increase duty cycle by
DC_DEC_VALUE
                addwf      PWM_DC
                btfsc      STATUS,C                  ; check if carry occurred
                goto       Turn_Off_PWM
                call       Load_DC_Register
                call       Display_Duty_Cycle
                goto       Send_Temp

Turn_Off_PWM
                bcf        sflag_event,pwm           ; clear pwm flag
                banksel    PWM_DC
                movlw      0xFF
                movwf      PWM_DC

                banksel    PORTC
                bsf        PORTC,2                   ; STPCLK# = 1, on
                banksel    CCP1CON
                clrf       CCP1CON                   ; PWM off

                goto       Send_Temp


;*****************************************************************
;* PWM setup
;*****************************************************************
Throttle

                banksel  sflag_event
                btfsc    sflag_event,pwm
                goto     Continue_PWM
                bsf      sflag_event,pwm
                banksel   PR2                       ; first time here, enable PWM
                movlw    0x3F                       ; 10 bit mode
                movwf    PR2                        ;set period
```

```
              banksel PWM_DC
             movlw    DC_INIT                ; Initialize Duty Cycle
             movwf    PWM_DC


             banksel  T2CON
             movlw    b'00000100'            ;enable timer2
             movwf    T2CON

             banksel  timer1count
             clrf     timer1count

             goto     Start_PWM
Continue_PWM

             banksel  timer1count

             btfss    timer1count,3          ; did timer1 roll over 8 times?
             goto     Send_Temp              ;return
                      clrf    timer1count

Start_PWM

             movlw    DC_DEC_VALUE           ; duty cycle decrement value
             banksel  PWM_DC
             subwf    PWM_DC,f               ; Duty Cycle = Duty Cycle - decrement
value
             btfss    STATUS,C               ; result negative
             clrf     PWM_DC                 ; if C=0.  Set DC to 0 %

             call     Load_DC_Register
             call     Display_Duty_Cycle

             goto     Send_Temp


;*****************************************************************
;                    MAIN LOOP BEGINS HERE
;*****************************************************************
Send_Command_Max1617
             banksel  INTCON
             bsf      INTCON,PEIE           ; enable peripheral interrupt
             bsf      INTCON,GIE            ; enable global interrupt
             banksel  sflag_event
Timer1_Overflow_Loop

             banksel  sflag_event
             btfss    sflag_event,tmr1_ovr
             goto     Timer1_Overflow_Loop
             bcf      sflag_event,tmr1_ovr

             banksel  Max1617_Cmd
             movf     Max1617_Cmd,w
             banksel  write_string
             movwf    write_string
             call     Init_Interrupt_Masks
             call     service_i2c           ; kick off start condition

Send_Command_Loop

             banksel  eflag_event          ; select SFR bank
             btfsc    eflag_event,ack_error; test for ack error event flag
             call     service_ackerror     ; service ack error
             banksel  sflag_event          ; select SFR bank
             btfss    sflag_event,rw_done  ; test if read/write cycle complete
             goto     Send_Command_Loop    ; goto main loop
```

```
              bcf       sflag_event,rw_done

                call      init_vars                 ; re-initialize variables
                banksel   INTCON
                bcf       INTCON,GIE                ;disable interrupts for display
                bcf       INTCON,PEIE

                return

Display_Max1617_Results

                movlw     high  (Display_Cmd_Jump + 1); fetch upper byte of jump table
address
                movwf     PCLATH                    ; load into upper PC latch

                banksel   Max1617_Cmd               ; recall command for jump table
lookup
                decf      Max1617_Cmd,w
                addlw     low (Display_Cmd_Jump + 1)
                btfsc     STATUS,C                  ; skip if carry didn't occur
                incf      PCLATH,f                  ; otherwise add carry

Display_Cmd_Jump
                movwf     PCL                      ; index into state machine jump table
                goto      Display_Remote_Temp      ; RRTE = 0x01
                goto      Display_Status_Reg       ; RSL = 0x02
                goto      Display_Configuration    ; RCL = 0x03
                goto      Display_Conversion       ; RCRA = 0x04 conversion rate
                goto      Display_Local_THIGH      ; RLHN = 0x05
                goto      Display_Local_TLOW       ; RLLI = 0x06
                goto      Display_Remote_THIGH     ; RRHI = 0x07
                goto      Display_Remote_TLOW      ; RRLS = 0x08
                goto      Display_Write_Config     ; WCA  = 0x09
                goto      Display_Write_Conv       ; WCRW = 0x0A
                goto      Display_Write_Local_THIGH   ; WLHO = 0x0B
                goto      Display_Write_Local_TLOW   ; WLLM = 0x0C
                goto      Display_Write_Remote_THIGH ; WRHA = 0x0D
                goto      Display_Write_Remote_TLOW  ; WRLN = 0x0E

;*****************************************************************
;* Display Temp
;*  This routine will output the ASCII chars to the USART which
;* will be displayed on the terminal.
;*****************************************************************
Display_Remote_Temp   ; RRTE = 0x01

                banksel   read_string
                movf      read_string,w
                banksel   twos_num
                movwf     twos_num
                call      convert_twos
                call      Display_Twos_Num
                movlw     0x0A
                call      Putc;linefeed

                return



;*************************************************************************
;* Load DC Register - This routine takes the 8-bit duty cycle value in PWM_DC
;*    and places it into the 10-bit DC registers.
;*    CCP1CON, CCP1X,CCP1Y are loaded with the two lsbs
;*    CCPR1L is loaded with the upper 6 bits.
;*
;*************************************************************************
```

```
Load_DC_Register
           banksel    CCP1CON
           movlw      b'00111100'
           iorwf      CCP1CON              ; pre-condition two LSBs CCP1X,CCP1Y
with 1's
                                           ; also set PWM mode
           banksel    PWM_DC
           movf       PWM_DC,w
           movwf      temp_hold

           btfss      PWM_DC,0             ; set or clear LSBs for duty cycle
           goto       clear_lsb0

check_lsb1
           banksel    PWM_DC
           btfss      PWM_DC,1
           goto       clear_lsb1
           goto       Load_DC_Register_Exit

clear_lsb0
           banksel    CCP1CON
           bcf        CCP1CON,4            ; clear lsb 0
           goto       check_lsb1
clear_lsb1
           banksel    CCP1CON
           bcf        CCP1CON,5

Load_DC_Register_Exit

           banksel    PWM_DC
           bcf        STATUS,C
           rrf        PWM_DC,f
           bcf        STATUS,C
           rrf        PWM_DC,w             ; rotate right to align with DC
register

           banksel    CCPR1L               ; load upper 6 bits of DC
           movwf      CCPR1L

           movf       temp_hold,w
           movwf      PWM_DC

           return                          ; return from Load_DC_Register


;*****************************************************************
;* Display Duty Cycle
;*   This routine converts the 8-bit duty cycle register to an ASCII
;*  decimal number.  Then it displays out out the RS232.
;*****************************************************************
Display_Duty_Cycle

             banksel  PWM_DC
             movf     PWM_DC,w
             banksel  twos_num            ; convert duty cycle to ASCII
             movwf    twos_num
             call     get_digits

             movlw    'D'                 ; Display duty cycle value in decimal
             call     Putc                ; absolute value not a percentage
             movlw    'C'
             call     Putc
             movlw    '='
             call     Putc
             banksel  hundreds_num
             movf     hundreds_num,w
             call     Putc
```

```
            banksel  tens_num
            movf     tens_num,w
            call     Putc
                     banksel  ones_num
            movf     ones_num,w
            call     Putc

            movlw    0x0a                ; print linefeed and CR
            call     Putc
            movlw    0x0d
            call     Putc


                     return              ; return from Display_Duty_Cycle


;*********************************************************************
;* Diplay twos complement number
;* INPUT:  sign_num = ASCII sign either + or -
;*         hundreds_num, tens_num, ones_num = ASCII of place
;* OUTPUT:  USART
;*********************************************************************
Display_Twos_Num
            movf     sign_num,w
            banksel  TXREG
            movwf    TXREG               ; copy the sign into TXREG
            banksel  TXSTA
            bsf      TXSTA,TXEN          ;enable transmitter


SIGN_XMIT
            btfss    TXSTA,TRMT
            goto     SIGN_XMIT

            banksel  hundreds_num
            movf     hundreds_num,w
            banksel  TXREG
            movwf    TXREG
            banksel  TXSTA
HUNDREDS_XMIT
            btfss    TXSTA,TRMT
            goto     HUNDREDS_XMIT

            banksel  tens_num
            movf     tens_num,w
            banksel  TXREG
            movwf    TXREG
            banksel  TXSTA
TENS_XMIT
            btfss     TXSTA,TRMT
            goto     TENS_XMIT

            banksel  ones_num
            movf     ones_num,w
            banksel  TXREG
            movwf    XREG
            banksel  TXSTA
ONES_XMIT
            btfss    TXSTA,TRMT
            goto     ONES_XMIT

            banksel  TXREG
            movlw    0x0D                ;return character
            movwf    TXREG
            banksel  TXSTA
CR_XMIT
```

```
            btfss     TXSTA,TRMT
            goto      CR_XMIT
            banksel   TXREG


            return

;*******************************************************************************
*****
Display_Status_Reg ; RSL = 0x02


            return

Display_Configuration    ; RCL = 0x03

            return

;*************************************************************************
;* Display Conversion Rate as the control byte read
;*************************************************************************
Display_Conversion ; RCRA = 0x04 conversion rate

            banksel  Str_High
            movlw    HIGH Conversion_String
            movwf    Str_High
            banksel  Str_Low
            movlw    LOW  Conversion_String
            movwf    Str_Low
            call     Puts


Get_Conversion_Rate_String
            movlw    high (Conversion_Jump +1)
            movwf    PCLATH
            banksel  read_string
            movf     read_string,w
            sublw    0x07                ; subtract w from literal
            btfsc    STATUS,C            ; if c=1 result is 0 or positive
            goto     Display_Conversion_OK
            movlw    0x08                ; print “reserved” if conversion rate
            goto     DC_Reserved         ; num returned is greater than 0x07

Display_Conversion_OK
            banksel  read_string
            movf     read_string,w
DC_Reserved
            addlw    low (Conversion_Jump + 1)


            btfsc    STATUS,C            ; skip if carry didn’t occur
            incf     PCLATH,f            ; otherwise add carry

Display_Conv_Jump
            movwf    PCL                 ; index into state machine jump table

Conversion_Jump
            movwf    PCL
            goto     P16seconds          ; 0
            goto     P8seconds           ; 1
            goto     P4seconds           ; 2
            goto     P2seconds           ; 3
            goto     P1second            ; 4
            goto     P0_5seconds         ; 5
            goto     P0_25seconds        ; 6
            goto     P0_125seconds       ; 7
            goto     Preserved           ; 8
```

```
P16seconds
            movlw   ‘1’
            call    Putc
            movlw   ‘6’
            call    Putc
            goto    Display_Conversion_Return

P8seconds
            movlw   ‘8’
            call    Putc
            goto    Display_Conversion_Return

P4seconds
            movlw   ‘4’
            call    Putc
            goto    Display_Conversion_Return
P2seconds
            movlw   ‘2’
            call    Putc
            goto    Display_Conversion_Return
P1second
            movlw   ‘1’
            call    Putc
            goto    Display_Conversion_Return
P0_5seconds
            movlw   ‘0’
            call    Putc
            movlw   ‘.’
            call    Putc
            movlw   ‘5’
            call    Putc
            goto    Display_Conversion_Return

P0_25seconds
            movlw    ‘0’
            call     Putc
            movlw    ‘.’
            call     Putc
            movlw    ‘2’
            call     Putc
            movlw   ‘5’
            call     Putc
            goto     Display_Conversion_Return
P0_125seconds
            movlw    ‘0’
            call                        Putc
            movlw   ‘.’
            call     Putc
            movlw    ‘1’
            call     Putc
            movlw    ‘2’
            call     Putc
            movlw    ‘5’
            call     Putc
            goto     Display_Conversion_Return

Preserved
            movlw    low (Reserved_String)
            banksel  Str_Low
            movwf    Str_Low
            movlw    high (Reserved_String)
            movwf    Str_High
            call     Puts                ; display “Reserved”


Display_Conversion_Return
```

```
                movlw    low (Seconds_String)
                banksel  Str_Low
                movwf    Str_Low
                movlw    high (Seconds_String)
                movwf    Str_High
                call      Puts               ; display “Seconds”

                movlw    0x0D
                call     Putc
                movlw     0x0A              ; xmit linefeed
                call     Putc

                return


Display_Local_THIGH ; RLHN = 0x05


                return

Display_Local_TLOW ; RLLI = 0x06

                return


;*************************************************************************
;* Display remote TLOW limit
;*************************************************************************
Display_Remote_THIGH     ; RRHI = 0x07

                banksel  Str_High
                movlw    high RTHIGH_String
                movwf    Str_High
                movlw    low  RTHIGH_String
                movwf    Str_Low
                call     Puts

                call     Display_Remote_Temp   ; Print out Remote TLOW limit
                movlw    0x0A
                call     Putc                  ; linefeed

                return

;*************************************************************************
;* Display remote TLOW limit
;*************************************************************************
Display_Remote_TLOW ; RRLS = 0x08

                banksel  Str_High
                movlw    high RTLOW_String
                movwf    Str_High
                movlw    low  RTLOW_String
                movwf    Str_Low
                call     Puts

                call     Display_Remote_Temp   ; Print out Remote TLOW limit
                movlw    0x0A
                call     Putc                  ; linefeed

                return

Display_Write_Config ; WCA  = 0x09


                return

Display_Write_Conv ; WCRW = 0x0A
```

```
            return

Display_Write_Local_THIGH  ; WLHO = 0x0B


            return

Display_Write_Local_TLOW   ; WLLM = 0x0C


            return

Display_Write_Remote_THIGH ; WRHA = 0x0D

            return

Display_Write_Remote_TLOW  ; WRLN = 0x0E

            return

;*****************************************************************
;* Initialize Interrupt Masks
;*****************************************************************

Init_Interrupt_Masks

            banksel   PIR1
            bcf      PIR1,SSPIF           ;clear SSPIF
            clrf     PIR1
            banksel  PIE1
            clrf     PIE1
            bsf      PIE1,SSPIE          ; enable SSP H/W interrupt

            banksel  PIE2                ; select SFR bank
            bsf      PIE2,BCLIE          ; enable bus collision interrupt

            banksel  PIE1
            bsf     PIE1,TMR1IE          ; enable timer1 interrupt

            return

;-----------------------------------------------------------------------
;   ************** Bus Collision Service Routine *****************
;-----------------------------------------------------------------------
service_buscoll
            banksel  i2cState            ; select GPR bank
            clrf     i2cState            ; reset I2C bus state variable
            call     init_vars           ; re-initialize variables
            banksel PORTB
            movlw   0xC0                  ;error code for bus collision
            movwf   PORTB

            return                       ;


;-----------------------------------------------------------------------
;   ************* Acknowledge Error Service Routine ***************
;-----------------------------------------------------------------------
service_ackerror
            banksel    eflag_event           ; select SFR bank
            bcf        eflag_event,ack_error  ; reset acknowledge error event flag
            clrf       i2cState              ; reset bus state variable
            call       init_vars             ; re-initialize variables
            banksel   PORTB
            movlw     0xE0                   ;error code for ACK
```

```
            movwf    PORTB

            return


;-----------------------------------------------------------------------
;   *****  INITIALIZE VARIABLES USED IN SERVICE_I2C FUNCTION  ******
;-----------------------------------------------------------------------
init_vars
            movlw    1                  ; byte count for max1617
            banksel  write_count        ; select GPR bank
            movwf    write_count        ; initialize write count
            movwf    read_count         ; initialize read count

            movlw    write_string       ; get write string array address
            movwf    write_ptr          ; initialize write pointer

            movlw    read_string        ; get read string placement address
            movwf    read_ptr           ; initialize read pointer

            movlw    ADDRESS            ; get address of slave
            movwf    temp_address       ; initialize temporary address hold reg

            banksel  eflag_event        ; select GPR bank
            clrf     eflag_event        ; initialize event flag variable
            btfss    sflag_event,pwm
            clrf     sflag_event        ; initialize event flag variable

            banksel  i2c_rw
            clrf     i2c_rw

            banksel  i2cState
            clrf     i2cState
            return


;*****************************************************************
;* Put string
;* INPUT: Str_Low, Str_High - points to low and high portion of
;*        string address.  ASCII string must be terminated
;*        with a null 0.  String must be in ROM with RETLWs
;* OUTPUT: To USART
;*****************************************************************
Puts

            banksel   TXSTA
            bsf      TXSTA,TXEN         ;enable transmitter
            banksel  index
            clrf     index
            banksel  Str_High           ; move high part of address
            movf     Str_High,w         ; of string into PCLATH
            movwf    PCLATH
Puts_lp

            banksel  TXSTA
Puts_trmt

            btfss     TXSTA,TRMT
            goto      Puts_trmt         ;no, loop


            call     Redirect_Call
            andlw    0xFF               ; test if null char of 0
            btfsc    STATUS,Z
            return
            banksel  TXREG
```

```
            movwf    TXREG               ;output character

            banksel  index
            incf     index,f

            goto      Puts_lp            ;no, loop
            return                       ;all displayed, finished
Redirect_Call
            banksel  Str_Low
            movf     Str_Low,w
            movwf    PCL                 ; goto string table

PutsError
            banksel  PORTB
            movlw    0xFE
            movwf    0xFE
            goto     $
;*****************************************************************
;* Put char
;* INPUT: w = character in ASCII to output
;* OUTPUT: To USART
;*****************************************************************
Putc
            banksel  TXREG
            movwf    TXREG
            banksel  TXSTA
putc_xmit
            btfss    TXSTA,TRMT
            goto     putc_xmit
            return


Debug1
            banksel  Str_High
            movlw    high Debug1_String
            movwf    Str_High
            movlw    low  Debug1_String
            movwf    Str_Low
            call     Puts
            return
Debug2
            banksel  Str_High
            movlw    high Debug2_String
            movwf    tr_High
            movlw    low  Debug2_String
            movwf    Str_Low
            call     Puts
            return


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

                     CODE   0x300
Welcome_Msg
                     banksel index
                     movf    index,w
                     addlw   low (Welcome_Msg  + 5)
                     movwf   PCL
                     DT                  “Clock throttling example using
MAX1617A and PIC16F873. rev 1.0”,0x0D,0x0A,0

;****************************************
;**** Conversion Periods *****************

Conversion_String
                     banksel index
                     movf    index,w
```

```
                        addlw   low (Conversion_String + 5)
                        movwf   PCL
                        DT                  “Conversion Period: “,0


Seconds_String
                        banksel index
                        movf    index,w
                        addlw   low (Seconds_String + 5)
                        movwf   PCL
                        DT                  “ seconds”,0

Reserved_String
                        banksel index
                        movf    index,w
                        addlw   low (Reserved_String + 5)
                        movwf   PCL
                        DT                  “Reserved”,0

RTLOW_String
                        banksel index
                        movf    index,w
                        addlw   low (RTLOW_String+ 5)
                        movwf   PCL
                        DT                  “Remote TLOW = “,0

RTHIGH_String
                        banksel index
                        movf    index,w
                        addlw   low (RTHIGH_String+ 5)
                        movwf   PCL
                                            DT“Remote THIGH = “,0

Debug1_String
                        banksel index
                        movf    index,w
                        addlw   low (Debug1_String + 5)
                        movwf   PCL
                        DT                  “Here 1”,0x0A,0x0D,0

Debug2_String
                        banksel index
                        movf    index,w
                        addlw   low (Debug2_String + 5)
                        movwf   PCL
                        DT                  “Here 2”,0x0a,0x0d,0


                                            END                                 ;
required directive
```

A.1 Init.asm

```
;*****************************************************************
;                                *
;   Implementing Master I2C with the MSSP module on a PICmicro          *
;                                *
;*****************************************************************
;                                                                      *
;  Filename:  init.asm                                                 *
;  Date:      04/04/2000                                               *
;  Revision: 1.00                                                      *
;                                                                      *
;  Tools:    MPLAB  5.00.00                                            *
;            MPLINK 2.00.11                                            *
;            MPASM  2.40.00                                            *
;                                                                      *
;                                                                      *
;                                                                      *
;*****************************************************************
;                                                                      *
;  Files required:                                                     *
;                                                                      *
;            init.asm                                                  *
;                                                                      *
;            p16f873.inc                                               *
;                                                                      *
;                                                                      *
;*****************************************************************
;                                                                      *
;  Notes:                                                              *
;                                                                      *
;                                                                      *
;*****************************************************************/

            #include <p16f873.inc> ; processor specific variable definitions
            errorlevel -302


                        GLOBAL init_ports    ; make function viewable for other modules

;-----------------------------------------------------------------------
; ****************** INITIALIZE PORTS ************************
;-----------------------------------------------------------------------
INIT_CODE             CODE

init_ports
            banksel   PORTA               ; select SFR bank
            clrf      PORTA               ;
            clrf      PORTB               ;

            movlw     0xff
            movwf     PORTC

            bsf       STATUS,RP0          ; select SFR bank
            movlw     b'00001111'         ;
            movwf     ADCON1              ;
            clrf      TRISB               ;
            clrf      TRISA               ;
            movlw     b'00011000'         ;RC4,C3 are SDA,SCL
            movwf     TRISC               ;RC2 is PWM1
            bcf       STATUS,RP0

            return

            END                           ; required directive
```

A.2 i2ccomm.asm

```
;*****************************************************************
;                                                                *
;                                                                *
;   Implementing Master I2C with the MSSP module on a PICmicro   *
;                                *
;*****************************************************************
;                                                                *
;  Filename: i2ccomm.asm                                         *
;  Date:     04/04/2000                                          *
;  Revision: 1.00                                                *
;                                                                *
;  Tools:    MPLAB  5.00.00                                      *
;            MPLINK 2.00.11                                      *
;            MPASM  2.40.00                                      *
;                                                                *
;*****************************************************************
;                                                                *
;  Files required:                                               *
;                                                                *
;             i2ccomm.asm                                        *
;                                                                *
;            i2ccomm.inc                                         *
;            flags.inc  (referenced in i2ccomm.inc file)         *
;            i2ccomm1.inc (must be included in main file)        *
;            p16f873.inc                                         *
;                                                                *
;*****************************************************************
;                                                                *
;  Notes:  The routines within this file are used to read from   *
;  and write to a Slave I2C device. The MSSP initialization      *
;  function is also contained within this file.                  *
;                                                                *
;*****************************************************************/

            #include <p16f873.inc>        ; processor specific definitions
            #include "i2ccomm.inc"        ; required include file
            errorlevel -302



#define FOSC          D’4000000’          ; define FOSC to PICmicro
#define I2CClock      D’400000’           ; define I2C bite rate
#define ClockValue (  ((FOSC/I2CClock)/4) -1) ;



;-----------------------------------------------------------------------
;  ********************** I2C Service ************************
;-----------------------------------------------------------------------
I2C_COMM     CODE
service_i2c
;            pagesel i2c_idle             ; ensure PCLATH is set for the section
;            call     i2c_idle            ; verify I2C idle check

             movlw    high I2CJump        ; fetch upper byte of jump table address
             movwf    PCLATH              ; load into upper PC latch
             movlw    i2cSizeMask
             banksel i2cState             ; select GPR bank
             andwf    i2cState,w          ; retrieve current I2C state
             addlw    low (I2CJump + 1)    ; calc state machine jump addr into W
             btfsc    STATUS,C            ; skip if carry occured
             incf     PCLATH,f            ; otherwise add carry
I2CJump      ; address where jump table branch occurs, this addr also used in fill
             movwf    PCL                  ; index into state machine jump table
```

```
                ; jump to processing for each state = i2cState value for each state

                goto     WrtStart            ; write start sequence          = 0
                goto     SendWrtAddr         ; write address, R/W=1          = 1
                goto     WrtAckTest          ; test ack,write data           = 2
                goto     WrtStop             ; do stop if done               = 3

                goto     ReadStart           ; write start sequence          = 4
                goto     SendReadAddr        ; write address, R/W=0          = 5
                goto     ReadAckTest         ; test acknowledge after address = 6
                goto     ReadData            ; read more data                = 7
                goto     ReadStop            ; generate stop sequence        = 8

I2CJumpEnd
    Fill (return), (I2CJump-I2CJumpEnd) + i2cSizeMask



;-----------------------------------------------------------------------
;  ******************** Write data to Slave  ********************
;-----------------------------------------------------------------------
; Generate I2C bus start condition        [ I2C STATE -> 0 ]
WrtStart
                banksel  write_ptr           ; select GPR bank
                movf     write_ptr,w         ; retrieve ptr address
                movwf    FSR                 ; initialize FSR for indirect access
                incf     i2cState,f          ; update I2C state variable
                banksel SSPCON2              ; select SFR bank
                bsf   SSPCON2,SEN            ; initiate I2C bus start condition

                return             ;

; Generate I2C address write (R/W=0)       [ I2C STATE -> 1 ]
SendWrtAddr
                banksel temp_address         ; select GPR bank
                bcf      STATUS,C            ; ensure carry bit is clear
                rlf      temp_address,w     ; compose 7-bit address
                incf     i2cState,f          ; update I2C state variable
                banksel  SSPBUF              ; select SFR bank
                movwf  SSPBUF                ; initiate I2C bus write condition
                return                       ;

; Test acknowledge after address and data write [ I2C STATE -> 2 ]
WrtAckTest
                banksel  SSPCON2             ; select SFR bank
                btfss    SSPCON2,ACKSTAT     ; test for acknowledge from slave
                goto     WrtData             ; go to write data module
                banksel  eflag_event         ; select GPR bank
                bsf      eflag_event,ack_error; set acknowledge error
                clrf     i2cState            ; reset I2C state variable
                banksel  SSPCON2             ; select SFR bank
                bsf   SSPCON2,PEN            ; initiate I2C bus stop condition
                return                       ;

; Generate I2C write data condition
WrtData
                movf     INDF,w              ; retrieve byte into w
                banksel  write_count         ; select GPR bank
                decfsz   write_count,f       ;test if all done with writes
                goto     send_byte           ; not end of string
                incf     i2cState,f          ; update I2C state variable
send_byte
                banksel  SSPBUF              ; select SFR bank
                movwf    SSPBUF              ; initiate I2C bus write condition
                incf     FSR,f               ; increment pointer
                return                       ;

; Generate I2C bus stop condition        [ I2C STATE -> 3 ]
```

```
WrtStop
            banksel  SSPCON2             ; select SFR bank
            btfss    SSPCON2,ACKSTAT     ; test for acknowledge from slave
            goto     no_error            ; bypass setting error flag
            banksel  eflag_event         ; select GPR bank
            bsf      eflag_event,ack_error; set acknowledge error
            clrf     i2cState            ; reset I2C state variable
            goto     write_or_read
no_error
            banksel   i2cState           ; select GPR bank
            incf     i2cState,f          ; update I2C state variable for read
write_or_read

            banksel i2c_rw
            btfss    i2c_rw,rw            ; read = 0, write = 1
            goto     ReadStart            ; if it is a read then continue to nxt
state
            btfsc    i2c_rw,stop
            goto     ReadStop
            bankse   Max1617_Data
            movf     Max1617_Data,w
            banksel  SSPBUF              ; select SFR bank
            movwf    SSPBUF              ; initiate I2C bus write condition
            banksel  i2cState
            movlw    0x03                ;switch to check ack after data write
            movwf    i2cState
            banksel  i2c_rw
            bsf      i2c_rw,stop          ;set up for stop on next state

            return

;-----------------------------------------------------------------------
;  ******************** Read data from Slave  ********************
;-----------------------------------------------------------------------
; Generate I2C start condition          [ I2C STATE -> 4 ]
ReadStart
            banksel  read_ptr            ; select GPR bank
            movf     read_ptr,W          ; retrieve ptr address
            movwf    FSR                 ; initialize FSR for indirect access
            incf     i2cState,f          ; update I2C state variable
            banksel   SSPCON2            ; select SFR bank
            bsf      SSPCON2,RSEN        ; initiate I2C bus REstart condition
            return             ;

; Generate I2C address write (R/W=1)       [ I2C STATE -> 5 ]
SendReadAddr
            banksel  temp_address        ; select GPR bank
            bsf      STATUS,C            ; ensure cary bit is clear
            rlf      temp_address,w      ; compose 7 bit address
            incf     i2cState,f          ; update I2C state variable
            banksel  SSPBUF              ; select SFR bank
            movwf  SSPBUF                ; initiate I2C bus write condition
            return             ;

; Test acknowledge after address write      [ I2C STATE -> 6 ]
ReadAckTest
            banksel  SSPCON2             ; select SFR bank
            btfss    SSPCON2,ACKSTAT     ; test for not acknowledge from slave
            goto     StartReadData       ; good ack, go issue bus read
            banksel  eflag_event         ; ack error, so select GPR bank
            bsf      eflag_event,ack_error; set ack error flag
            clrf     i2cState            ; reset I2C state variable
            banksel  SSPCON2             ; select SFR bank
            bsf      SSPCON2,PEN         ; initiate I2C bus stop condition
            return

StartReadData
```

```
            bsf     SSPCON2,RCEN        ; generate receive condition
            banksel i2cState            ; select GPR bank
            incf    i2cState,f          ; update I2C state variable
            return

; Read slave I2C                [ I2C STATE -> 7 ]
ReadData
            banksel SSPBUF               ; select SFR bank
            movf    SSPBUF,w             ; save off byte into W
            banksel read_count           ; select GPR bank
            decfsz  read_count,f         ; test if all done with reads
            goto    SendReadAck          ; not end of string so send ACK


; Send Not Acknowledge
SendReadNack
            movwf   INDF                 ; save off null character
            incf    i2cState,f           ; update I2C state variable

            banksel SSPCON2              ; select SFR bank
            bsf     SSPCON2,ACKDT        ; acknowledge bit state to send (not ack)
            bsf     SSPCON2,ACKEN        ;initiate acknowledge sequence
            return

; Send Acknowledge
SendReadAck
            movwf    INDF                ; no, save off byte
            incf    FSR,f                ; update receive pointer

            banksel SSPCON2             ; select SFR bank
            bcf     SSPCON2,ACKDT        ; acknowledge bit state to send
            bsf     SSPCON2,ACKEN        ; initiate acknowledge sequence
            btfsc   SSPCON2,ACKEN        ; ack cycle complete?
            goto    $-1                  ; no, so loop again
            bsf     SSPCON2,RCEN         ; generate receive condition
            return              ;


; Generate I2C stop condition          [ I2C STATE -> 8 ]
ReadStop
            banksel SSPCON2              ; select SFR bank
            bcf     PIE1,SSPIE           ; disable SSP interrupt
            bsf     SSPCON2,PEN          ; initiate I2C bus stop condition
            banksel i2cState             ; select GPR bank
            clrf    i2cState             ; reset I2C state variable
            bsf     sflag_event,rw_done  ; set read/write done flag

            return



;-----------------------------------------------------------------------
;  ****************** Generic bus idle check **********************
;-----------------------------------------------------------------------
; test for i2c bus idle state
i2c_idle
            banksel SSPSTAT             ; select SFR bank
            btfsc    SSPSTAT,R_W        ; test if transmit is progress
            goto     $-1                ; module busy so wait
            banksel SSPCON2             ; select SFR bank
            movf    SSPCON2,w           ; get copy of SSPCON2 for status bits
            andlw   0x1F                ; mask out non-status bits
            btfss   STATUS,Z            ; test for zero state, if Z set, bus is idle
            goto    $-3                 ; bus is busy so test again
            return                      ; return to calling routine
```

```
;-----------------------------------------------------------------------
; ****************** INITIALIZE MSSP MODULE ******************
;-----------------------------------------------------------------------

init_i2c
            banksel  SSPADD              ; select SFR bank
            movlw    ClockValue          ; read selected baud rate
            movwf    SSPADD              ; initialize I2C baud rate

            bsf      SSPSTAT,6           ; select I2C input levels for SMBUS
            bcf      SSPSTAT,7           ; enable slew rate

;           movlw    b'00011000'         ;
;           iorwf    TRISC,f             ; ensure SDA and SCL are inputs
;           movwf    TRISC
            bcf      STATUS,RP0          ; select SFR bank
            movlw    b'00111000'         ;
            movwf    SSPCON              ; Master mode, SSP enable
            return                       ; return from subroutine


            END                          ; required directive
```

A.3 mastri2c.inc

```
;*****************************************************************
;                                                                *
;  Filename: mastri2c.inc                                        *
;  Date:     04/01/2000                                          *
;  Revision: 1.00                                                *
;                                                                *
;  Tools:    MPLAB  5.00.00                                      *
;            MPLINK 2.00.11                                      *
;            MPASM  2.40.00                                      *
;                                                                *
;*****************************************************************


;*******  INTERRUPT CONTEXT SAVE/RESTORE VARIABLES
INT_VAR       UDATA     0x20               ; create uninitialized data "udata" section
w_temp        RES       1        ;
status_temp   RES       1        ;
pclath_temp   RES       1


INT_VAR1    UDATA  0xA0                    ; reserve location 0xA0
w_temp1     RES   1


;*******  GENERAL PURPOSE VARIABLES
GPR_DATA              UDATA
temp_hold     RES   1                      ; temp variable for string compare
ptr1          RES   1                      ; used as pointer in string compare
ptr2          RES   1                      ; used as pointer in string compare
count         RES   1
index         RES   1
sign_num      RES   1
hundreds_num  RES   1
tens_num      RES   1
ones_num      RES   1
twos_num      RES   1
timer1count   RES   1
```

```
Max1617_Cmd  RES    1
Str_High     RES    1
Str_Low      RES    1
PWM_DC       RES    1


STRING_DATA                              UDATA
write_string RES       D'30'
read_string  RES       D'30'


             EXTERN    init_ports           ; reference linkage for function
```

A.4 i2ccomm.inc

```
;*****************************************************************
;                                                                *
;  Filename: i2ccomm.inc                                         *
;  Date:     04/04/2000                                          *
;  Revision: 1.00                                                *
;                                                                *
;  Tools:    MPLAB  5.00.00                                      *
;            MPLINK 2.00.11                                      *
;            MPASM  2.40.00                                      *
;                                                                *
;*****************************************************************
;                                                                *
;  Notes:                                                        *
;                                                                *
;            This file is to be included in the i2ccomm.asm file *
;                                                                *
;                                                                *
;*****************************************************************/

             #include "flags.inc"          ; required include file

i2cSizeMask            EQU                 0x0F


             GLOBAL     sflag_event         ; make variable viewable for other modules
             GLOBAL     eflag_event         ; make variable viewable for other modules
             GLOBAL     i2cState            ; make variable viewable for other modules
             GLOBAL     read_count          ; make variable viewable for other modules
             GLOBAL     write_count         ; make variable viewable for other modules
             GLOBAL     write_ptr           ; make variable viewable for other modules
             GLOBAL     read_ptr            ; make variable viewable for other modules
             GLOBAL     temp_address        ; make variable viewable for other module
             GLOBAL     init_i2c            ; make function viewable for other modules
             GLOBAL     service_i2c         ; make function viewable for other modules
             GLOBAL     i2c_rw
             GLOBAL     Max1617_Data


;*******  GENERAL PURPOSE VARIABLES
GPR_DATA               UDATA
sflag_event      RES     1                  ; variable for i2c general status flags
eflag_event      RES     1                  ; variable for i2c error status flags
i2cState         RES     1                  ; I2C state machine variable
read_count       RES     1                  ; variable used for slave read byte count
write_count      RES     1                  ; variable used for slave write byte count
write_ptr        RES     1                  ; variable used for pointer (writes to)
read_ptr         RES     1                  ; variable used for pointer (reads from)
temp_address     RES     1                  ; variable used for passing address to
functions
```

```
i2c_rw          RES     1
Max1617_Data RES   1

;*********************************************************************
;                                                                    *
;  Additional notes on variable usage:                               *
;                                                                    *
;  The variables listed below are used within the function           *
;  service_i2c. These variables must be initialized with the         *
;  appropriate data from within the calling file. In this            *
;  application code the main file is 'mastri2c.asm'. This file       *
;  contains the function call to service_i2c. It also contains       *
;  the function for initializing these variables, called 'init_vars' *
;                                                                    *
;  To use the service_i2c function to read from and write to an      *
;  I2C slave device, information is passed to this function via      *
;  the following variables.                                          *
;                                                                    *
;                                                                    *
;  The following variables are used as function parameters:          *
;                                                                    *
;  read_count  - Initialize this variable for the number of bytes    *
;         to read from the slave I2C device.                         *
;  write_count - Initialize this variable for the number of bytes    *
;         to write to the slave I2C device.                          *
;  write_ptr  - Initialize this variable with the address of the     *
;         data string to write to the slave I2C device.              *
;  read_ptr   - Initialize this variable with the address of the     *
;         location for storing data read from the slave I2C          *
;         device.                                                    *
;  temp_address - Initialize this variable with the address of the   *
;         slave I2C device to communicate with.                      *
;                                                                    *
;                                                                    *
;  The following variables are used as status or error events        *
;                                                                    *
;  sflag_event - This variable is implemented for status or          *
;         event flags. The flags are defined in the file             *
;         'flags.inc'.                                               *
;  eflag_event - This variable is implemented for error flags. The   *
;         flags are defined in the file 'flags.inc'.                 *
;                                                                    *
;                                                                    *
;  The following variable is used in the state machine jump table.   *
;                                                                    *
;  i2cState   - This variable holds the next I2C state to execute.   *
;                                                                    *
;*********************************************************************

i2ccomm1.inc
;*********************************************************************
;                                                                    *
;  Filename: i2ccomm1.inc                                            *
;  Date:     04/04/2000                                              *
;  Revision: 1.00                                                    *
;                                                                    *
;  Tools:    MPLAB  5.00.00                                          *
;            MPLINK 2.00.11                                          *
;            MPASM  2.40.00                                          *
;                                                                    *
;*********************************************************************
;                                                                    *
;  Notes:                                                            *
;                                                                    *
;            This file is to be included in the <main.asm> file. The *
;  <main.asm> notation represents the file which has the             *
;  subroutine calls for the functions 'service_i2c' and 'init_i2c'.  *
```

```
                                                                        *
;                                                                       *
;                                                                       *
;***********************************************************************/

            #include "flags.inc"          ; required include file


            GLOBAL    write_string        ; make variable viewable for other modules
            GLOBAL    read_string         ; make variable viewable for other modules

            EXTERN    sflag_event         ; reference linkage for variable
            EXTERN    eflag_event         ; reference linkage for variable
            EXTERN    i2cState            ; reference linkage for variable
            EXTERN    read_count          ; reference linkage for variable
            EXTERN    write_count         ; reference linkage for variable
            EXTERN    write_ptr           ; reference linkage for variable
            EXTERN    read_ptr            ; reference linkage for variable
            EXTERN    temp_address        ; reference linkage for variable

            EXTERN    init_i2c            ; reference linkage for function
            EXTERN    service_i2c         ; reference linkage for function
            EXTERN    i2c_rw
            EXTERN    Max1617_Data

;***********************************************************************
;                                                                       *
;  Additional notes on variables declared as EXTERN                     *
;                                                                       *
;  The variables listed below are used within the function              *
;  service_i2c. These variables must be initialized with the            *
;  appropriate data from within the calling file. In this               *
;  application code the main file is 'mastri2c.asm'. This file          *
;  contains the function call to service_i2c. It also contains          *
;  the function for initializing these variables, called 'init_vars'    *
;                                                                       *
;  To use the service_i2c function to read from and write to an         *
;  I2C slave device, information is passed to this function via         *
;  the following variables.                                             *
;                                                                       *
;                                                                       *
;  The following variables are used as function parameters:             *
;                                                                       *
;  read_count  - Initialize this variable for the number of bytes       *
;         to read from the slave I2C device.                            *
;  write_count - Initialize this variable for the number of bytes       *
;         to write to the slave I2C device.                             *
;  write_ptr  - Initialize this variable with the address of the        *
;         data string to write to the slave I2C device.                 *
;  read_ptr   - Initialize this variable with the address of the        *
;         location for storing data read from the slave I2C             *
;         device.                                                       *
;  temp_address - Initialize this variable with the address of the      *
;         slave I2C device to communicate with.                         *
;                                                                       *
;                                                                       *
;  The following variables are used as status or error events           *
;                                                                       *
;  sflag_event - This variable is implemented for status or             *
;         event flags. The flags are defined in the file                *
;         'flags.inc'.                                                  *
;  eflag_event - This variable is implemented for error flags. The      *
;         flags are defined in the file 'flags.inc'.                    *
;                                                                       *
;                                                                       *
;  The following variable is used in the state machine jumnp table.     *
;                                                                       *
```

```
;  i2cState   - This variable holds the next I2C state to execute.         *
;                                                                          *
;**************************************************************************
```

A.5 max1617a.inc

```
;Max1617A constants
;4/24/00 - Rick Evans

NUMSTATES      EQU       10
RRTE           EQU       0x01                ;Command for reading remote temp sensor
RSL            EQU       0x02                ;Command for reading status reg
RCL            EQU       0X03                ;Command for reading configuration byte
RCRA           EQU       0X04                ;Command for reading conversion rate byte
RLHN           EQU       0X05                ;Command for reading local THIGH limit
RLLI           EQU       0X06                ;Command for reading local TLOW limit
RRHI           EQU       0X07                ;Command for reading remote thigh limit
RRLS           EQU       0X08                ;Command for reading remote TLOW limit
WCA            EQU       0x09                ;Command for writing configuration reg
WCRW           EQU       0x0A                ;Command for writing conversion rate reg
WLHO           EQU       0x0B                ;Command for writing local THIGH limit
WLLM           EQU       0x0C                ;Write local TLOW limit
WRHA           EQU       0x0D                ;Command for writing remote THIGH limit reg
WRLN           EQU       0x0E                ;Command for writing remote TLOW limit reg
```

A.6 flags.inc

```
;**************************************************************************
;                                                                          *
;  Filename: flags.inc                                                     *
;  Date:     04/04/2000                                                    *
;  Revision: 1.00                                                          *
;                                                                          *
;  Tools:    MPLAB  5.00.00                                                *
;            MPLINK 2.00.11                                                *
;            MPASM  2.40.00                                                *
;                                                                          *
;**************************************************************************
;                                                                          *
;  Notes:                                                                  *
;                                                                          *
;            This file defines the flags used in the i2ccomm.asm file.     *
;                                                                          *
;                                                                          *
;**************************************************************************/


; bits for variable sflag_event
#define sh1            0          ; place holder
#define sh2            1          ; place holder
#define sh3            2          ; place holder
#define sh4            3          ; place holder
#define sh5            4          ; place holder
#define pwm            5          ; if pwm is enabled (clk throttling)
#define tmr1_ovr       6          ; timer1 overflow
#define rw_done        7          ; flag bit


; bits for variable eflag_event
#define ack_error      0          ; flag bit
#define eh1            1          ; place holder
#define eh2            2          ; place holder
#define eh3            3          ; place holder
```

```
#define eh4           4          ; place holder
#define eh5           5          ; place holder
#define eh6           6          ; place holder
#define eh7           7          ; place holder


; bits for variable i2c_rw
#define rw            0          ; flag bit
#define stop          1          ; stop flag bit for write data command
#define rw2           2          ; place holder
#define rw3           3          ; place holder
#define rw4           4          ; place holder
#define rw5           5          ; place holder
#define rw6           6          ; place holder
#define rw7           7          ; place holder
```