TEXAS INSTRUMENTS

MSP430 SMBus

Application Report

September 1999

Mixed Signal Products

SLAA073

Contents

MSP430 SMBus

Richard Baker

ABSTRACT

This application report describes a software implementation of the system management bus (SMBus) for the MSP430 microcontroller. It includes all master protocols, an interrupt-driven slave, and master usage examples. SMBus is derived from the I2C and is commonly used in smart batteries and other system devices.

1 Introduction

The SMbus is a two-wire synchronous serial interface derived from the I^2C. Like the I^2C, it uses a serial clock (SCL) and a serial data line (SDA). The lines are kept from floating with pullup resistors or a current source, and devices connected to the bus must have an open-drain or open-collector connection. This results in the bus having the wired AND of all devices connected. This means that any single device can drive the bus low. Devices can only drive the bus low or release it. Since devices connected to the bus may operate at different voltages, they cannot drive the bus high.

Devices may be categorized as masters or slaves. A master can initiate a transmission and provide clock signals for the transmission. A slave can receive or send data, but the transmission must be started by a master. Devices can have characteristics common to both master and slave, or be strictly one or the other. For example, a slave can occasionally become a master and send data alerting the system to a critical condition. The application dictates how each device should perform.

Each device on the bus has a 7-bit address. This allows for up to 128 devices to be connected, although some addresses are reserved. Different commercially-available devices have addresses assigned by the SMBus address coordinating committee.

Communication uses seven transfer protocols. They are Quick Command, Send Byte, Receive Byte, Write Byte/Word, Read Byte/Word, Process Call, and Block Read/Write. Each protocol has its own exact series of steps, but all protocols follow a similar outline.

Communication is always initiated with a start condition (S). The start condition begins with both lines high, followed by SDA going low and then SCL going low. After a start condition, communication consists of 8-bit blocks followed by an acknowledge (ACK). An ACK is generated by the slave driving SDA low while the master clocks SCL. If SDA is not pulled low during the SCL pulse, it is a not-acknowledged (NACK). The first 8-bit block is the 7-bit slave address followed by one bit which denotes transfer direction in all protocols except Quick Command. In the Quick Command protocol, this bit is the actual command. After address and direction, the master expects an acknowledge from the slave. It pulses the SCL line once and watches for the slave to pull SDA low during the pulse. If the ACK is not received, the master must abort the transmission and repeat the transfer later. This feature can also be used to check if a slave is present on the bus. After the acknowledge, either another 8-bit block can be sent/received, or a stop condition (P) can be sent. The stop condition begins with both lines low, followed by SCL going high and then SDA going high. The remaining 8-bit blocks can be data, commands, or another address and read/write bit. The exact format of each protocol can be found in the System Management Bus specification (http://www.sbs–forum.org/smbus/index.html).

Since the bus uses the wired AND of all signals on the bus, there is an inherent arbitration mechanism. If more than one master tries to put data onto the bus, the first one to release the bus while another drives it low loses arbitration. For this to be effective, the masters must check the value of the bus every time the bus is released when generating a high signal. If the bus is ever low while a master has released it, the master must abort communication and try again later. As a result of this arbitration scheme, low addresses have priority over high addresses. If two masters begin a transfer at the same time, the one sending the higher address is the first to release the bus while the other holds it low, thus losing arbitration.

The speed of the bus is 10 kHz to 100 kHz. This is similar to the I^2C, except for the minimum speed: I^2C has no specified minimum speed, so devices can take as long as they want. SMBus adds more restrictions by adding a minimum speed. Even though there is a minimum speed, slave devices are not restricted to run as fast as the master. The only requirement is that they operate at a minimum of 10 kHz. The variable speed feature is implemented as follows: if a slave needs extra time to complete an operation, it can force the master to wait by holding SCL low. This is referred to as clock-low extending. The master must wait for the slave to release SCL, but will time out if the slave takes longer than the specification allows. This is done to maintain the minimum speed.

More SMBus information can be found at the Smart Battery System Implementers Forum (SBS IF). The SBS IF is a committee made up of representatives from Benchmarq Microelectronics Inc., Duracell Inc., Energizer Power Systems, Intel Corporation, Linear Technology Corporation, Maxim Integrated Products, Mitsubishi Electric Corporation, National Semiconductor Corporation, Toshiba Battery Co., and Varta Batterie AG. The committee publishes the SMBus specification, which is available in pdf format from the website.

2 Hardware Connections

Figure 1 shows an example of the SMBus connections. Here the MSP430 is connected to a smart battery. The smart battery is used to power the bus and the MSP430, but this is not necessary as long as there is a common ground connection. R_P are pull up resistors. To keep power consumption low, the largest possible R_P value which meets the rise time requirements should be used. There are also SMBus driver chips that take the place of the pullup resistors. These are designed to provide maximum current when the bus is rising, and minimum current when it is low.

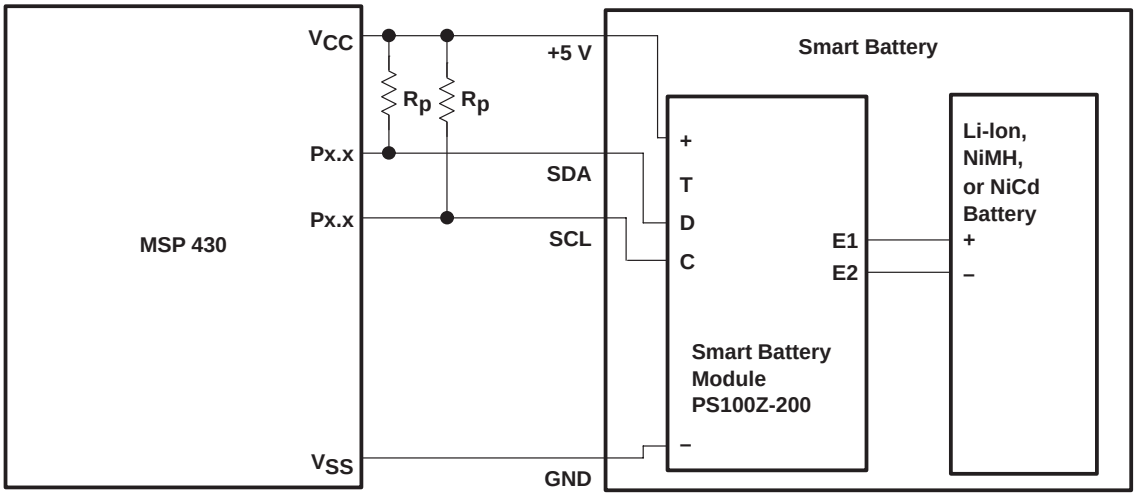

Figure 1. SMBus Connections Example

NOTE: The unused T connection is a thermistor. It is used in Li-Ion battery packs as an extra-safety measure in the event of a smart battery module or SMBus failure.

3 SMBus Master Code

The SMBus master code for the MSP430 implements master functionality for all protocols. Parameters are passed on the stack and registers are preserved. The maximum size of the stack is 40 bytes. Memory beyond the stack is unaffected, except during block operations. During a block transfer, the beginning of the block (it will grow toward upper memory addresses) is passed as a parameter. During a block write, the size of the block is known before the transfer, but during a block read the size is sent by the slave. Care should be taken to make sure the block will not overwrite program or data code.

The program has two main parts: high level subroutines and low level subroutines. The low level subroutines handle the bit shifts, timing, and port operations. The SMBus protocols are essentially made up of several building blocks that are called and repeated in a certain order. The low-level subroutines handle these building blocks. The high-level routines protect the registers and provide a user interface. There is one for each protocol and they set up parameters and call the low-level subroutines.

Two device pins are used to implement the open-collector requirement of the SMBus by switching between input and output mode. The port-output bits for the pins are always set to 0. This means that the MSP430 can release the bus by switching the pin to input, or drive it low by switching the pin to output. The pins are selected by equates at the beginning of the program; these are SDA, SCL, and DNC. SDA and SCL correspond to the SMBus data and clock lines. The values determine which bits of the port will be used for each. DNC is both SCL and SDA. The port is selected by the equates OUT, DIR, and IN.

Depending on the application, it is likely that not all protocols need to be used. The code size can be reduced by commenting out the unused routines. Removing high-level subroutines is relatively straightforward, but it is also possible to carefully remove some low-level routines.

Since the SMBus is both multimaster and hot-pluggable, errors can occur. These can result from the bus being busy, a slave not being present, competition from another master, or noise on the bus. The master code uses error routines and an error flag to handle errors. When an error is detected, the transfer is aborted and the program sets an error flag and returns from the high level subroutine. The user program must check the error flag to determine if it is successful (returns a 1) or unsuccessful (returns a 0). The user program can then decide how to handle the error.

The basic program provides the low and high-level routines in a memory space separate from the user code. A blank spot is provided for the user program. To transfer data on the SMBus, simply call the appropriate high level subroutine. Examples of parameter passing are provided in the comments above each subroutine. The master code can be found in Appendix A. Example programs are also provided in the appendixes to illustrate various protocols and smart battery system applications.

4 Master Code Examples

4.1 Address Hunter

The first example is fairly simple. It searches for devices on the bus and displays the address of the last device found. This is very useful when testing bus connections. It was originally written to find device addresses. It is a good example of reducing the master code size by removing unused subroutines. Since it is fairly simple, a large amount of the code is removed. It is currently set up to run from the monitor on an MSP430x33x evaluation kit (EVK part number MSP-EVK430x330).

4.2 Smart Battery Data

This example is much larger than the first one. It uses the read-word, write-word, and read-block protocols to communicate with a smart battery. It receives the pack temperature, pack voltage, manufacturer, chemistry, and design capacity. It also writes a value to the RemainingCapacityAlarm register and reads it back out. These operations are done in a loop and displayed on the LCD. It shows how to use the more complex protocols and how to process some of the data that can be returned. It is currently configured to run on an MSP430x33x evaluation kit (EVK Part Number MSP-EVK430x330) with the SMBus routines in EPROM and the user program running in the monitor.

5 SMBus Slave Code

This code implements an interrupt-driven SMBus slave. The *.equ* statements at the beginning of the program can select the slave's address, pin, and port. It implements the send-byte protocol (which for a slave means the master sends a byte to the slave). In between interrupts, the slave can either sleep or do other work. Since the interrupt will occur whenever a high-to-low transition occurs on SDA, the slave will spend most of its time in the interrupt routine if the bus is busy. To keep the slave from watching the bus even when other slaves are being addressed, external hardware can be used which watches the bus and causes an interrupt only when the correct address is seen. Another pitfall of using interrupts without external hardware is the time required to process an interrupt. Since it takes several cycles, it is possible to miss start conditions. A polling interpretation is less likely to miss a start, but is always busy watching the bus. Which approach is better depends on the application. The interrupt version allows the processor to do other work when the bus is idle, but may not get a transmission on the first attempt. The polling version does not allow any other work to be done whether the bus is idle or not. It also uses more power, since the processor cannot be placed into a low-power mode while the bus is idle.

Other transmission protocols can be derived from this one. This protocol handles start and stop conditions, and can receive 7-bit addresses and 8-bit data/ command blocks. By manipulating the different sections in the correct order, several other protocols can be handled.

The registers are not pushed onto the stack in order to conform to the fastest possible start condition in the spec. The time between SDA falling and SCL falling can be as low as 4 ms. If the master is known to take longer, it may be possible to push the registers onto the stack at the beginning of the routine, but this can result in missed start conditions with a fast master.

6 Conclusion

The desire to improve the run time of battery-powered devices has lead to sophisticated battery chemistries as well as battery monitoring technology. The SMBus provides a well-defined, robust interface to these smart battery devices, and the ultra-low-power operation of the MSP430 makes it an excellent fit for these systems.

7 Reference

[1] SBS Implementers Forum, *System Management Bus Specification*, Version 1.1, 12/11/98.

Appendix A SMBus Master Software

```
;*******************************************************************
; SMBus master program  with parameter passing on the stack
;*******************************************************************
RAM_orig  .set   00240h      ; Free Memory startadress
SP_orig   .set   005DEh      ; stackpointer
EPROM     .set   08850h      ; EPROM location
;--- Control register definitions
WDTCTL     .equ   0120h
WDTHold    .equ   80h
WDT_wrkey  .equ   05A00h
;PORT 0
P0IE       .equ   015h
P0DIR      .equ   012h
P0IN       .equ   010h
P0OUT      .equ   011h
; port 2
P2IN       .equ   028h
P2OUT      .equ   029h
P2IE       .equ   02Dh
P2SEL      .equ   02Eh
P2DIR      .equ   02Ah
SDA        .equ   020h
SCL        .equ   080h
DNC        .equ   0A0h
OUT        .equ   P2OU
TDIR       .equ   P2DIR
IN         .equ   P2IN
LCD1       .equ   031h
LCDM       .equ   030h
IE1        .equ   0h
IE2        .equ   01h
IFG1       .equ   02h
IFG2       .equ   03h
block      .equ   0550h
alcd       .equ   0B500h      ; location of ASCII LCD table
;*******************************************************************
; Reset : Initialize processor
;*******************************************************************
          .sect ”MAIN”,RAM_orig
RESET
          MOV    #SP_orig,SP                  ; initialize stackpointer
;*******************************************************************
```

```
; User program begining
;***********************************************************************
; user program goes here
;***********************************************************************
; User program end
;***********************************************************************
.sect     ”smbus”, EPROM
;***********************************************************************
; high level subroutines, call low level subroutines to implement
; individual protocols
;***********************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; quick command protocol  1
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step       Direction
;                     -------------------------
;                     start      out
;                     address    out
;                     R/W            out
;                     acknowledge in
;                     stop           out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   unused
;   unused
;   unused
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h        ; error code space
;        PUSH   #00Bh       ; push the address onto the stack
;        SUB    #06h, SP    ; other parameters unused
;        CALL   #qcp
;        ADD    #08h, SP    ; deallocate parameter space
;        POP    R9          ; pop the error code
;        CMP    #00h, R9
;        JZ     error       ;  unsuccessful – run error routine
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
qcp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  28(SP), R7       ; get the address from the stack
          CALL   #sbit            ; send start and address
          ;CALL  #sendone         ; send a 1  or
          CALL    #sendzero        ; send a 0
          CALL   #ack             ; wait for ACK
          CALL   #pbit            ; send stop
                    MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send byte protocol   2
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
```

```
;                     acknowledge   in
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   unused
;   unused
;   data
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h        ; error code space
;        PUSH   #00Bh       ; push the address onto the stack
;        PUSH   #055h       ; data to be sent
;        SUB    #04h, SP    ; other parameters unused
;        CALL   #sbp
;        ADD    #08h, SP    ; deallocate parameter space
;        POP    R9          ; pop the error code
;        CMP    #00h, R9
;        JZ     error       ;  unsuccessful – run error routine
;
sbp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  28(SP), R7       ; get the address from the stack
          MOV.B  26(SP), R8       ; get the data from the stack
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command byte
          CALL   #ack             ; wait for ACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
```

```
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; receive byte protocol  3
  ; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     ---------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     data          in
;                     NACK          out
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   data
;   unused
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h        ; error code space
;        PUSH   #00Bh       ; push the address onto the stack
;        PUSH   #08h        ; command code
;        SUB    #04h, SP    ; allocate space for return value
;        CALL   #rcbp
;        POP    R8          ; pop data to R8
```

```
;        ADD    #06h, SP    ; deallocate parameter space
;        POP    R9          ; pop the error code
;        CMP    #00h, R9
;        JZ     error       ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rbp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send a 1 for read
          CALL   #ack             ; wait for ACK
          CALL   #rbyte           ; receive data byte
          CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  R14, 22(SP)      ; copy the data byte to the stack
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
; read word protocol  7
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step       Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     data          in
;                     acknowledge   in
;                     data          in
;                     acknowledge   in
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   low data byte
;   high data byte
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #008h         ; push the command for battery temperature
;        PUSH   #00h          ; reserve a byte for data
;        PUSH   #00h          ; reserve a byte for data
;        CALL   #rwp
;        POP    R10           ; pop data to R10
;        POP    R11           ; pop data to R11
;        ADD    #04h, SP      ; free the address and command space
;        POP    R9            ; pop the error code
```

```
;        CMP    #00h, R9
;        JZ     error          ;  unsuccessful – run error routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rwp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send a 1 for read
          CALL   #ack             ; wait for ACK
          CALL   #rbyte           ; receive low data byte
          MOV.B  R14, 22(SP)      ; store low data byte on stack
          CALL   #sack            ; send an  ACK
          CALL   #rbyte           ; receive high data byte
          CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  R14, 24(SP)      ; store high data byte on stack
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
```

```
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; block write protocol  8
; ========================
; Registers affected: none; Memory affected   : maximum of 40 bytes on stack, memory block
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     byte count    out
;                     acknowledge   in
;                     data          out
;                     acknowledge   in
;                      :
;                     repeat  data, acknowledge n times
;                      :
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   byte count
;   block pointer
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #0020h        ; push the command for battery temperature
;        PUSH   #00550h       ; pointer to begining of block
;        PUSH   #04h          ; reserve a byte for byte count
;        CALL   #blkw
;        POP    R10           ; pop count to R10
;        ADD    #06h, SP
;        POP    R9            ; pop the error code
;        CMP    #00h, R9
```

```
;        JZ     error          ;  unsuccessful – run error routine

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
blkw
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          MOV    24(SP), R9       ; block start address
          MOV.B  #00h, R15        ; counter to loop through bytes
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          MOV.B  22(SP), R8       ; copy the byte count
          CALL   #sbyte           ; send the byte count
          CALL   #ack             ; wait for ACK
blk_rep   MOV.B  0(R9), R8        ; move the data from the block to data
          CALL   #sbyte           ; send data byte
          CALL   #ack             ; wait for ACK
          INC    R15              ; increment block position counter
          INC    R9               ; increment the block pointer
          CMP    22(SP), R15      ; check for end of data
          JNZ    blk_rep          ; repeat until end
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
```

```
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; block read protocol  9
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack, memory block
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     W             out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     start         out
;                     address       out
;                     R             out
;                     acknowledge   in
;                     byte count    in
;                     acknowledge   out
;                     data          in
;                     acknowledge   out
;                      :
;                     repeat  data, acknowledge n times
;                      :
;                     nack          out
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   byte count
;   block pointer
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
```

```
;        PUSH   #0020h        ; push the command for battery temperature
;        PUSH   #00550h       ; pointer to begining of block
;        PUSH   #00h          ; reserve a byte for byte count
;        CALL   #blkr
;        POP    R10           ; pop count to R10
;        ADD    #06h, SP
;        POP    R9        ; pop the error code
;        CMP    #00h, R9
;        JZ     error     ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
blkr
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  #00h, R15        ; counter to loop through bytes
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          MOV    24(SP), R9       ; block start address
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send 1 for read
          CALL   #ack             ; wait for ack
          CALL   #rbyte
          MOV.B  R14, 22(SP)      ; push the count onto the stack
          CALL   #sack            ; send an ack
rblk_rep  CALL   #rbyte           ; receive data byte
          MOV.B  R14, 0(R9)       ; move the data from the block to data
          INC    R15              ; increment block position counter
          INC    R9               ; increment the block pointer
          CMP    22(SP), R15      ; check for end of data
          JZ     blk_done         ; repeat until end
          CALL   #sack            ; send an ACK
```

```
          CMP    22(SP), R15      ; check for end of data
          JNZ    rblk_rep         ; repeat until end
blk_done  CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
;*******************************************************************
; low level subroutines, implement common parts of all transfers
;*******************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send start condition and an address
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbit    ; release both lines
          BIC.B   #DNC, DIR
        ; check for free bus
          MOV     #05h, R10     ; counter for ticking off 50 micro sec
wait
          MOV.B   IN, R11       ; copy the inputs to R11
          AND     #DNC, R11     ; mask all but inputs
          CMP     #DNC, R11     ; if either SDA or SCL is low the bus is busy
          JNZ     busy
          DEC     R10           ; decrement the counter
          JNZ     wait
rsbit     BIC.B   #DNC, DIR     ; redundant line release so that a repeated start can use
                                ; the same routine
        ; send out the start condition
          BIS.B   #SDA, DIR
          BIS.B   #SCL, DIR
        ; send out the address
          MOV.B   #07h, R13     ; counter for the 7 bit address
          MOV.B   R7, R11       ; copy the address to R11
ashift    RLA.B   R11           ; rotate left so MSB of 7 bit address is in position 7
          MOV.B   R11, R12      ; copy so it can be masked without loss of data
          AND.B   #080h, R12    ; mask all but MSB
          CMP.B   #00h, R12     ; compare it to 0
```

```
          JNZ     one
          CALL    #sendzero     ; send a one
          JMP     zero
one       CALL    #sendone      ; send a zero
zero      DEC     R13           ; decrement the 7 counter
          JNZ     ashift        ; if <7 bits have been sent repeat
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; No ACK received case, do a STOP then busy
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbusy     CALL    #pbit          ; generate a stop, then  busy code
          ADD    #02h, SP        ; remove stuff from stack since RET wasn't used
          MOV.B  #00h, 28(SP)    ; error code
          POP    R13             ; restore registers
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Wait for acknowledge
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ack
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          ;MOV     #008h, R11    ; counter for approx. 50 micro sec
          MOV     #02FFh, R11    ; counter extended PS100Z-200 clock low extends here
clkex     MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy           ;arbitration--took too long to rise
          AND.B   #SCL, R10      ; check SCL
          JZ      clkex
        ; wait for the ack signal (SDA going low)
          MOV     #05F6h, R11    ; counter to timeout for a NACK
wack
          MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      sbusy          ; timeout NACK-- send P bit and retry
          AND.B   #SDA, R10      ; mask all but SDA
          JNZ     wack           ; either see the ACK or wait here for timeout
```

```
          BIS.B   #SCL, DIR      ; put 0 on SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Transmit stop condition
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pbit
          BIS.B   #SDA, DIR      ; put a 0 on SDA
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          NOP
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          NOP
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a byte (uses sendzero and sendone)
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbyte
        ; send out the data
          MOV.B   #08h, R13      ; counter for approx. 50 micro sec.
          MOV.B   R8, R11        ; copy the data to R11
dshift
          MOV.B   R11, R12
          AND.B   #080h, R12     ; mask all but MSB
          CMP.B   #00h, R12      ; compare it to 0
          JNZ     on
          CALL    #sendzero      ; send a one
          JMP     zer
on        CALL    #sendone       ; send a zero
zer
          RLA.B   R11            ; rotate left so data to go out is in MSB
          DEC     R13            ; decrement the 8 counter
          JNZ     dshift
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; receive a byte
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rbyte
          MOV.B   #08h,  R12     ; counter for 8 data bits
          BIC.B   #SDA, DIR      ; put a 1 on SDA
rrep
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          MOV     #0035h, R13    ; counter extended PS100Z–200 clock low extends here
clkxtn    MOV.B   IN, R10        ; check the bus
          DEC     R13
          JZ      busy           ; if it took too long to rise abort
```

```
          AND.B   #SCL, R10
          JZ      clkxtn          ; if SCL is not high wait
          MOV.B   IN, R10         ; check the bus
          RLA.B   R11             ; move over for input as LSB
          AND.B   #SDA, R10       ; check SDA
          JZ      inzero
          BIS.B   #01h, R11       ; set the LSB
inzero    DEC     R12
          JNZ     rrep            ; repeat for the rest of the byte
          MOV.B   R11, R14        ; copy the data to the variable
          BIS.B   #SCL, DIR       ; put a 0 on SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send a NACK
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
nack
          BIC.B   #SDA, DIR       ; put a 1 on SDA
          BIS.B   #SCL, DIR       ; put a 0 on SCL
          BIC.B   #SCL, DIR       ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
        ; wait for the SCL line to rise
clkn      MOV.B   IN, R10
          AND.B   #SCL, R10       ; check SCL
          JZ      clkn
          BIS.B   #SCL, DIR       ; release SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send a ACK
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sack      ; wait for SDA release by slave
          MOV     #000Fh, R11     ; counter for approx. 50 micro sec
swaita    MOV.B   IN, R10         ; check the bus
          DEC     R11
          JZ      busy
          AND.B   #SDA, R10       ; check SCL
          JZ      swaita
          BIS.B   #SDA, DIR       ; pull SDA low
          BIC.B   #SCL, DIR       ; put a 1 on SCL
          MOV     #000Fh, R11     ; counter for approx. 50 micro sec
swait     MOV.B   IN, R10         ; check the bus
          DEC     R11
          JZ      busy
          AND.B   #SCL, R10       ; check SCL
          JZ      swait
          BIS.B   #SCL, DIR       ; put a 0 on SCL
          BIC.B   #SDA, DIR       ; release SDA
```

```
          RET
;--- clear LCD
show_clr
          MOV    #15, r5         ; clear display memory
show_clr1
          MOV.b  #0, LCD1-1(r5)
          DEC    r5
          JNZ    show_clr1
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; LCD  Definitions
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
LCD_TYPE
;---STK/EVK LCD
a       .equ     01h
b       .equ     02h
c       .equ     10h
d       .equ     04h
e       .equ     80h
f       .equ     20h
g       .equ     08h
h       .equ     40h
;--- character definitions

LCD_Tab .byte  a+b+c+d+e+f              ; displays "0"
        .byte   b+c                     ; displays "1"
        .byte   a+b+d+e+g               ; displays "2"
        .byte   a+b+c+d+g               ; displays "3"
        .byte   b+c+f+g                 ; displays "4"
        .byte   a+c+d+f+g               ; displays "5"
        .byte   a+c+d+e+f+g             ; displays "6"
        .byte   a+b+c                   ; displays "7"
        .byte   a+b+c+d+e+f+g           ; displays "8"
        .byte   a+b+c+d+f+g             ; displays "9"
        .byte   0                       ; displays ":" blank
        .byte   g                       ; displays ";" -
        .byte   a+d+e+f                 ; displays "<" [
        .byte   d+g                     ; displays "="
        .byte   a+b+c+d                 ; displays ">" ]
        .byte   a+b+e+g                 ; displays "?"
        .byte   a+b+d+e+f+g             ; displays "@"
        .byte   a+b+c+e+f+g             ; displays "A"
        .byte   c+d+e+f+g               ; displays "B" b
        .byte   a+d+e+f                 ; displays "C"
        .byte   b+c+d+e+g               ; displays "D" d
        .byte   a+d+e+f+g               ; displays "E"
```

```
        .byte   a+e+f+g                 ; displays ”F”
        .byte   a+b+c+d+f+g             ; displays ”G”
        .byte   b+c+e+f+g               ; displays ”H”
        .byte   b+c                     ; displays ”I”
        .byte   b+c+d+e                 ; displays ”J”
        .byte   0                       ; displays ”K”
        .byte   d+e+f                   ; displays ”L”
        .byte   a+b+c+e+f               ; displays ”M”
        .byte   c+e+g                   ; displays ”N” n
        .byte   c+d+e+g                 ; displays ”O” o
        .byte   a+b+e+f+g               ; displays ”P”
        .byte   0                       ; displays ”Q”
        .byte   e+g                     ; displays ”R” r
        .byte   a+c+d+f+g               ; displays ”S”
        .byte   d+e+f+g                 ; displays ”T” t
        .byte   c+d+e                   ; displays ”U” u
        .byte   0                       ; displays ”V”
        .byte   0                       ; displays ”W”
        .byte   0                       ; displays ”X”
        .byte   b+c+d+f+g               ; displays ”Y”
        .byte   a+b+d+e+g               ; displays ”Z” 2
LCD_Tab_End
        .even
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Interrupt vectors
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          ; .sect     ”Int_Vect”,0FFFFh–1   ; without monitor
          ; .word     RESET                 ; POR, ext. Reset, Watchdog
          ; .end
           .sect     ”Int_Vect”,05FFh–1    ; with monitor
           .word     RESET                 ; POR, ext. Reset, Watchdog
           .end
```

Appendix B Address Hunter Software

```
;*******************************************************************
; Address Hunter program
;-----------------------------------
;    This program scans the SMBus for connected devices.  It starts at
; address 0x00 and goes through the 128 possible device addresses.  The
; address currently being tested is displayed on the right side of the LCD
; and the address of the last device found is shown on the left.  When devices
; are found the program pauses, moves the address to the left, and then continues.
;
;    Device presence is tested by sending a start condition, the device address,
; and then checking for an acknowledge.  Then the next address is tested.
;
;    The address hunter is an example of removing unused portions of code.
; Since it doesn't need to use any complete protocols, the high level routines
; and some low level routines have been removed.
;*******************************************************************
RAM_orig  .set   00240h      ; Free Memory startadress
SP_orig   .set   005DEh      ; stackpointer
;--- Control register definitions
WDTCTL     .equ    0120h
WDTHold    .equ    80h
WDT_wrkey  .equ    05A00h
;PORT 0
P0IE       .equ    015h
P0DIR      .equ    012h
P0IN       .equ    010h
P0OUT      .equ    011h
; port 2
P2IN       .equ    028h
P2OUT      .equ    029h
P2IE       .equ    02Dh
P2SEL      .equ    02Eh
P2DIR      .equ    02Ah
SDA        .equ    020h
SCL        .equ    080h
DNC        .equ    0A0h
OUT        .equ    P2OUT
DIR        .equ    P2DIR
IN         .equ    P2IN
LCD1       .equ    031h
LCDM       .equ    030h
IE1        .equ    0h
IE2        .equ    01h
IFG1       .equ    02h
IFG2       .equ    03h
```

```
address    .equ    0500h               ; RAM address for tested variable

;-----------------------------------------------------------------------
; Reset : Initialize processor
;-----------------------------------------------------------------------
          .sect ”MAIN”,RAM_orig
RESET
          MOV    #SP_orig,SP                 ; initialize stackpointer
;-----------------------------------------------------------------------
; User program begining
;-----------------------------------------------------------------------
        CALL     #show_clr
; Reset the address
res MOV.B    #00FFh, address      ; start the address at 0xFF
; begining of loop through the addresses
retry
        INC.B  address               ; move to the next address, will make 0xFF into 0x00
        CMP.B  #080h, address        ; check if address is 80
        JZ     res                   ; reset the address if it is too high
; delay
        MOV    #02Fh, R7             ; nested delay loop
dela2   MOV    #031Fh, R6
dela    DEC    R6
        JNZ    dela
        DEC    R7
        JNZ    dela2
; display the address about to be tested
        MOV    #0000h, R5            ; clear (make sure upper byte is cleared)
        MOV.B  address, R5           ; move R5 to lower byte
        AND    #000Fh, R5            ; mask off upper nibble
        MOV.B  LCD_Tab(R5),LCD1+1    ; display the lower nibble 0–F
        MOV.B  address, R5           ; move R5 to lower byte
        AND    #00F0h, R5            ; mask off lower nibble
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        MOV.B  LCD_Tab(R5), LCD1+2   ; display the upper nibble 0–F

        PUSH   #00h          ; error code space
        PUSH   address       ; push the address onto the stack
        SUB    #06, SP
        CALL   #hunt
        POP    R10           ; pop data to R10
        POP    R11           ; pop data to R11
        ADD    #04h, SP      ; free the address and command space
```

```
        POP    R9             ; pop the error code
        CMP    #00h, R9
        JZ     retry              ; retry if unsuccessful

        MOV     #0FFh, R7
dela4   MOV     #0B4Fh, R6
dela3   DEC     R6
        JNZ     dela3
        DEC     R7
        JNZ     dela4
        MOV    #0000h, R5            ; clear (make sure upper byte is cleared)
        MOV.B  address, R5           ; move R5 to lower byte
        AND    #000Fh, R5            ; mask off upper nibble
        MOV.B  LCD_Tab(R5),LCD1+4    ; display the lower nibble 0–F
        MOV.B  address, R5           ; move R5 to lower byte
        AND    #00F0h, R5            ; mask off lower nibble
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        RRA    R5                    ; shift right
        MOV.B  LCD_Tab(R5), LCD1+5   ; display the upper nibble 0–F
        JMP     retry               ; repeat for the next address

;***********************************************************************
; User program end
;***********************************************************************
;***********************************************************************
; high level subroutines, call low level subroutines to implement
; individual protocols
;***********************************************************************
;***********************************************************************
; Hunt is a modified portion of read byte protocol
;***********************************************************************
hunt      PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
```

```
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

;***********************************************************************
; low level subroutines, implement common parts of all transfers
;***********************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send start condition and an address
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbit    ; release both lines
          BIC.B   #DNC, DIR
        ; check for free bus
          MOV     #05h, R10     ; counter for ticking off 50 micro sec
wait
          MOV.B   IN, R11       ; copy the inputs to R11
          AND     #DNC, R11     ; mask all but inputs
          CMP     #DNC, R11     ; if either SDA or SCL is low the bus is busy
          JNZ     busy
          DEC     R10           ; decrement the counter
          JNZ     wait
rsbit     BIC.B   #DNC, DIR     ; redundant line release so that a repeated start can use
                                ;the  same routine
        ; send out the start condition
          BIS.B   #SDA, DIR
          BIS.B   #SCL, DIR
        ; send out the address
          MOV.B   #07h, R13     ; counter for the 7 bit address
          MOV.B   R7, R11       ; copy the address to R11
ashift    RLA.B   R11           ; rotate left so MSB of 7 bit address is in position 7
          MOV.B   R11, R12      ; copy so it can be masked without loss of data
          AND.B   #080h, R12    ; mask all but MSB
          CMP.B   #00h, R12     ; compare it to 0
          JNZ     one
          CALL    #sendzero     ; send a one
```

```
          JMP     zero
one       CALL    #sendone      ; send a zero
zero      DEC     R13           ; decrement the 7 counter
          JNZ     ashift        ; if <7 bits have been sent repeat
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; No ACK received case, do a STOP then busy
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbusy     CALL   #pbit          ; generate a stop, then  busy code
          ADD    #02h, SP       ; remove stuff from stack since RET wasn't used
          MOV.B  #00h, 28(SP)   ; error code
          POP    R13            ; restore registers
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Handle a busy system
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
busy3     ADD     #06h, SP      ; remove extra data from sendone or sendzero
busy      ADD     #02h, SP      ; remove data from stack since RET wasn't used

          MOV.B  #00h, 28(SP)   ; error code
          POP    R13            ; restore registers
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a 1
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sendone   PUSH    R10
          PUSH    R11
```

```
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          MOV.B   #04h, R11      ; adjust number of loops for correct time
clkext    MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy3          ; took too long for it to rise--arbitration
          AND.B   #SCL, R10      ; check SCL
          JZ      clkext
       ; arbitration:  make sure that SDA is really high, if not arbitration
       ;was lost to another master on the bus
          MOV.B   IN, R10        ; check the bus
          AND.B   #SDA, R10      ; mask all but SDA
          JZ      busy3
          BIS.B   #SCL, DIR      ; release SCL
          MOV.B   IN, R10        ; detect a repeated start condition
          AND.B   #SDA, R10
      JZ       busy3
          POP     R11
          POP     R10
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a 0
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sendzero
          PUSH    R10
          PUSH    R11
          BIS.B   #SDA, DIR      ; put a 0 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          MOV     #04h, R11      ; counter to time the rise of SCL
clke      MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy3          ; arbitration-- took too long for SCL to rise
          AND.B   #SCL, R10      ; check SCL
          JZ      clke
          BIS.B   #SCL, DIR      ; release SCL
          POP     R11
          POP     R10
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Wait for acknowledge
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ack
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
```

```
        ; due to clock low extending, don't outrun the slave
          ;MOV     #008h, R11    ; counter for approx. 50 micro sec
          MOV     #02FFh, R11    ; counter extended PS100Z-200 clock low extends here
clkex     MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy           ;arbitration--took too long to rise
          AND.B   #SCL, R10      ; check SCL
          JZ      clkex
        ; wait for the ack signal (SDA going low)
          MOV     #05F6h, R11    ; counter to timeout for a NACK
wack
          MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      sbusy          ; timeout NACK-- send P bit and retry
          AND.B   #SDA, R10      ; mask all but SDA
          JNZ     wack           ; either see the ACK or wait here for timeout
          BIS.B   #SCL, DIR      ; put 0 on SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Transmit stop condition
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pbit
          BIS.B   #SDA, DIR      ; put a 0 on SDA
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          NOP
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          NOP
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          RET
;--- clear LCD
show_clr
          MOV    #15, r5         ; clear display memory
show_clr1
          MOV.b  #0, LCD1-1(r5)
          DEC    r5
          JNZ    show_clr1
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; LCD  Definitions
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

LCD_TYPE
;---STK/EVK LCD
a       .equ     01h
b       .equ     02h
c       .equ     10h
```

```
d       .equ     04h
e       .equ     80h
f       .equ     20h
g       .equ     08h
h       .equ     40h
;--- character definitions

LCD_Tab .byte  a+b+c+d+e+f              ; displays ”0”
        .byte   b+c                     ; displays ”1”
        .byte   a+b+d+e+g               ; displays ”2”
        .byte   a+b+c+d+g               ; displays ”3”
        .byte   b+c+f+g                 ; displays ”4”
        .byte   a+c+d+f+g               ; displays ”5”
        .byte   a+c+d+e+f+g             ; displays ”6”
        .byte   a+b+c                   ; displays ”7”
        .byte   a+b+c+d+e+f+g           ; displays ”8”
        .byte   a+b+c+d+f+g             ; displays ”9”
        .byte   a+b+c+e+f+g             ; displays ”A”
        .byte   c+d+e+f+g               ; displays ”B” b
        .byte   a+d+e+f                 ; displays ”C”
        .byte   b+c+d+e+g               ; displays ”D” d
        .byte   a+d+e+f+g               ; displays ”E”
        .byte   a+e+f+g                 ; displays ”F”
LCD_Tab_End
        .even
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Interrupt vectors
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

           .sect     ”Int_Vect”,05FFh-1    ; with monitor
           .word     RESET                 ; POR, ext. Reset, Watchdog
           .end
```

Appendix C Smart Battery Data Software

```
;*****************************************************************
; SBData Program
; ------------------------------
; Reads the Temperature, Voltage, ManufacturerName, DeviceChemistry, Capacity, and
; RemainingCapacityAlarm from a SMBus battery monitor.  It also writes to the
; RemainingCapacityAlarm register.
;
; This particular application was tested using a PowerSmart PS100Z–200 smart
; battery module.  Other battery data can be read by changing the command.
; PowerSmart smart battery chips support SMBus V1.0 and the Smart Battery
; data spec.
; It is currently configured to run with the user program under the monitor
; on a 337 EVK and the SMBus routines in EPROM.  An ASCII LCD table is also
; needed.   The file used here is ASCII.txt.  It is pointed to by alcd.
;*****************************************************************
RAM_orig  .set   00240h      ; Free Memory startadress
SP_orig   .set   005DEh      ; stackpointer
EPROM     .set   08850h      ; EPROM location
;--- Control register definitions
WDTCTL          .equ    0120h
WDTHold         .equ    80h
WDT_wrkey       .equ    05A00h
;PORT 0 stuff
P0IE        .equ    015h
P0DIR       .equ    012h
P0IN        .equ    010h
P0OUT       .equ    011h
; port 2 stuff
P2IN        .equ    028h
P2OUT       .equ    029h
P2IE        .equ    02Dh
P2SEL           .equ    02Eh
P2DIR           .equ    02Ah
SDA     .equ    020h
SCL             .equ    080h
DNC             .equ    0A0h
OUT             .equ    P2OUT
DIR             .equ    P2DIR
IN              .equ    P2IN
LCD1            .equ    031h
LCDM            .equ    030h
IE1             .equ    0h
IE2             .equ    01h
IFG1            .equ    02h
IFG2            .equ    03h
```

```
block           .equ    0550h
alcd            .equ    0B500h      ; location of ASCII LCD table

;***********************************************************************
; Reset : Initialize processor
;***********************************************************************
          .sect ”MAIN”,RAM_orig
RESET
          MOV    #SP_orig,SP                  ; initialize stackpointer
;***********************************************************************
; User program begining
;***********************************************************************

repeat
        CMP.B   #0FFh, R8
        JNZ     skp
        INC.B   R7            ; increment the upper byte write value for the write word
        MOV.B   #00h, R8
skp
        INC.B   R8            ; increment the write value for the write word protocol

skp0
        ; read the Temperature
        CALL    #show_clr
        PUSH    #00h          ; error code space
        PUSH    #000Bh        ; push the address onto the stack
        PUSH    #008h         ; push the command for battery temperature
        PUSH    #00h          ; reserve a byte for data
        PUSH    #00h          ; reserve a byte for data
        CALL    #rwp
        POP     R10           ; pop data to R10
        POP     R11           ; pop data to R11
        ADD     #04h, SP      ; free the address and command space
        POP     R9            ; pop the error code
        CMP     #00h, R9
        JZ      skp0          ; retry if unsuccessful
        RLA     R11
        RLA     R11
        RLA     R11
        RLA     R11
        RLA     R11
        RLA     R11
        RLA     R11
        RLA     R11
        XOR     R10, R11       ; combine the two data bytes
        MOV     R11, R12
```

```
        SUB    #0AAAh, R12     ; convert to Celsius
        CALL   #display        ; call display routine
        XOR.B  #40h, LCD1+1       ; add a decimal point
        MOV.B  alcd+'T', LCD1+5
        CALL   #delay             ; call delay routine
skp1    ; read the Voltage
        CALL   #show_clr
        PUSH   #00h           ; error code space
        PUSH   #000Bh         ; push the address onto the stack
        PUSH   #009h          ; push the command for battery voltage
        PUSH   #00h           ; reserve a byte for data
        PUSH   #00h           ; reserve a byte for data
        CALL   #rwp
        POP    R10            ; pop data to R10
        POP    R11            ; pop data to R11
        ADD    #04h, SP       ; free the address and command space
        POP    R9             ; pop the error code
        CMP    #00h, R9
        JZ     skp1           ; retry if unsuccessful
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        XOR    R10, R11         ; combine the two data bytes
        MOV    R11, R12
        CALL   #display          ; call display routine
        XOR.B  #40h, LCD1+3       ; add a decimal point
        MOV.B  alcd+'U', LCD1+5
        CALL   #delay         ; call delay routine
skp2    ; read the ManufacturerName
        CALL   #show_clr
        PUSH   #00h           ; error code space
        PUSH   #000Bh         ; push the address onto the stack
        PUSH   #0020h         ; push the command for battery temperature
        PUSH   #block         ; reserve a byte for data
        PUSH   #00h           ; reserve a byte for data
        CALL   #blkr
        POP    R10            ; pop data to R10
        ADD    #06h, SP
        POP    R9             ; pop the error code
        CMP    #00h, R9
        JZ     skp2           ; retry if unsuccessful
```

```
        MOV.B  block, R5             ; copy a letter
        MOV.B  alcd(R5), LCD1+6      ; display a letter
        MOV.B  block+1, R5           ; copy a letter
        MOV.B  alcd(R5), LCD1+5   ; display a letter
        MOV.B  block+2, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+4      ; display a letter
        MOV.B  block+3, R5           ; copy a letter
        MOV.B  alcd(R5), LCD1+3   ; display a letter
        MOV.B  block+4, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+2      ; display a letter
        MOV.B  block+5, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+1   ; display a letter
        MOV.B  block+6, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+0   ; display a letter
        CALL   #delay             ; call delay routine
skp3    ; read the DeviceChemistry
        CALL   #show_clr
        PUSH   #00h           ; error code space
        PUSH   #000Bh         ; push the address onto the stack
        PUSH   #0022h         ; push the command for battery temperature
        PUSH   #block         ; reserve a byte for data
        PUSH   #00h           ; reserve a byte for data
        CALL   #blkr
        POP    R10            ; pop data to R10
        ADD    #06h, SP
        POP    R9             ; pop the error code
        CMP    #00h, R9
        JZ     skp3           ; retry if unsuccessful
        MOV.B  block, R5          ; copy a letter
        MOV.B  alcd(R5), LCD1+6   ; display a letter
        MOV.B  block+1, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+5   ; display a letter
        MOV.B  block+2, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+4   ; display a letter
        MOV.B  block+3, R5        ; copy a letter
        MOV.B  alcd(R5), LCD1+3   ; display a letter
        CALL   #delay             ; call delay routine
skp4    ; read the DesignCapacity
        CALL   #show_clr
        PUSH   #00h       ; error code space
        PUSH   #000Bh     ; push the address onto the stack
        PUSH   #0018h     ; push the command for battery voltage
        PUSH   #00h       ; reserve a byte for data
        PUSH   #00h       ; reserve a byte for data
        CALL   #rwp
        POP    R10        ; pop data to R10
```

```
        POP    R11         ; pop data to R11
        ADD    #04h, SP    ; free the address and command space
        POP    R9          ; pop the error code
        CMP    #00h, R9
        JZ     skp4            ; retry if unsuccessful
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        XOR    R10, R11
        MOV    R11, R12
        CALL   #display        ; call display routine
        XOR.B  #40h, LCD1+3        ; add a decimal point
        MOV.B  alcd+'A', LCD1+1
        MOV.B  alcd+'H', LCD1+0
        CALL   #delay          ; call delay routine
skp5    ; write the RemainingCapacityAlarm
        CALL   #show_clr
        PUSH   #00h        ; error code space
        PUSH   #000Bh      ; push the address onto the stack
        PUSH   #001h       ; push the command for battery voltage
        PUSH   R8          ; reserve a byte for data
        PUSH   R7          ; reserve a byte for data
        CALL   #wwp
        POP    R11         ; pop data to R10
        POP    R10         ; pop data to R11
        ADD    #04h, SP    ; free the address and command space
        POP    R9          ; pop the error code
        CMP    #00h, R9
        JZ     skp5        ; retry if unsuccessful
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        XOR    R10, R11
        MOV    R11, R12
        CALL   #display    ; call display routine
        MOV.B  alcd+'S', LCD1+6
```

```
        CALL   #delay     ; call delay routine
skp6    ; read the RemainingCapacityAlarm
        CALL   #show_clr
        PUSH   #00h         ; error code space
        PUSH   #000Bh       ; push the address onto the stack
        PUSH   #001h        ; push the command for battery voltage
        PUSH   #00h         ; reserve a byte for data
        PUSH   #00h         ; reserve a byte for data
        CALL   #rwp
        POP    R10          ; pop data to R10
        POP    R11          ; pop data to R11
        ADD    #04h, SP     ; free the address and command space
        POP    R9           ; pop the error code
        CMP    #00h, R9
        JZ     skp6         ; retry if unsuccessful
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        RLA    R11
        XOR    R10, R11
        MOV    R11, R12
        CALL   #display   ; call display routine
        MOV.B  alcd+'R', LCD1+6
        CALL   #delay     ; call delay routine
        JMP    repeat     ; end of program endless loop

;-----------------------------------------------------------------------
; User program end
;-----------------------------------------------------------------------
delay
         PUSH    R6
         PUSH    R7
         MOV     #00DFh, R7       ; nested loop delay routine
dela4    MOV     #0D40h, R6
dela3    DEC     R6
         JNZ     dela3
         DEC     R7
         JNZ     dela4
         POP     R7
         POP     R6
         RET
```

```
display                     ; display routine
        PUSH    R5
        PUSH    R12
        PUSH    R13
        PUSH    R14
        PUSH    R15
        MOV     #16, R15                ; hex to BCD conversion routine
        CLR     R14                     ; takes R12 as input, output to R13 and R14
        CLR     R13
ls1     RLA     R12
        DADD    R13, R13
        DADD    R14, R14
        DEC     R15
        JNZ     ls1
        MOV.B   R13, R5                 ; move 2 lower BCD digits to R5
        AND     #000Fh, R5              ; mask all but lower nibble
        MOV.B   LCD_Tab(R5),LCD1+0      ; display lower nibble
        MOV.B   R13, R5                 ; move 2 lower BCD digits to R5
        AND     #00F0h, R5              ; mask all but upper nibble
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        MOV.B   LCD_Tab(R5), LCD1+1     ; display upper nibble
        SWPB    R13                     ; swap upper and lower bytes in R13
        MOV.B   R13, R5                 ; move 2 upper BCD digits to R5
        AND     #000Fh, R5              ; mask all but lower nibble
        MOV.B   LCD_Tab(R5),LCD1+2      ; display lower nibble
        MOV.B   R13, R5                 ; move 2 upper BCD digits to R5
        AND     #00F0h, R5              ; mask all but upper nibble
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        MOV.B   LCD_Tab(R5), LCD1+3     ; display upper nibble
        MOV.B   R14, R5                 ; move 2 lower BCD digits to R5
        AND     #000Fh, R5              ; mask all but lower nibble
        MOV.B   LCD_Tab(R5),LCD1+4      ; display lower nibble
        MOV.B   R14, R5                 ; move 2 lower BCD digits to R5
        AND     #00F0h, R5              ; mask all but upper nibble
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        RRA     R5                      ; right shift
        MOV.B   LCD_Tab(R5), LCD1+5     ; display upper nibble
        POP     R15
```

```
        POP    R14
        POP    R13
        POP    R12
        POP    R5
        RET
           .sect     ”smbus”, EPROM

;***********************************************************************
; high level subroutines, call low level subroutines to implement
; individual protocols
;***********************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; quick command protocol  1
; =========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   unused
;   unused
;   unused
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h        ; error code space
;        PUSH   #00Bh       ; push the address onto the stack
;        SUB    #06h, SP    ; other parameters unused
;        CALL   #qcp
;        ADD    #08h, SP    ; deallocate parameter space
;        POP    R9          ; pop the error code
;        CMP    #00h, R9
;        JZ     error       ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
qcp
         PUSH   SR
```

```
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  28(SP), R7       ; get the address from the stack
          CALL   #sbit            ; send start and address
          ;CALL  #sendone         ; send a 1  or
          CALL    #sendzero       ; send a 0
          CALL   #ack             ; wait for ACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; write byte protocol   4
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;             command       out
;                     acknowledge   in
;                     data          out
;                     acknowledge   in
;                     stop          out
;
```

```
;
; Stack parameters
; ----------------
;   error_flag        low mem
;   unused
;   data
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h        ; error code space
;        PUSH   #00Bh       ; push the address onto the stack
;        PUSH   #01h        ; push the command onto the stack
;        PUSH   #055h       ; data to be sent
;        SUB    #02h, SP    ; other parameters unused
;        CALL   #wbp
;        ADD    #08h, SP    ; deallocate parameter space
;        POP    R9          ; pop the error code
;        CMP    #00h, R9
;        JZ     error       ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
wbp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          MOV.B  24(SP), R8       ; data to write
          CALL   #sbyte           ; send data byte
          CALL   #ack             ; wait for ACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
```

```
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; write word protocol   5
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     --------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     data          out
;                     acknowledge   in
;                     data          out
;                     acknowledge   in
;                     stop          out
;
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   unused
;   data
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h         ; error code space
;        PUSH   #00Bh        ; push the address onto the stack
;        PUSH   #01h         ; command code
;        PUSH   #0AAh        ; data to be sent low byte
;        PUSH   #055h        ; data to be sent high byte
;        CALL   #wwp
;        ADD    #08h, SP     ; deallocate parameter space
```

```
;        POP    R9           ; pop the error code
;        CMP    #00h, R9
;        JZ     error        ;  unsuccessful – run error routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
wwp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          MOV.B  24(SP), R8       ; data to write –low byte
          CALL   #sbyte           ; send low data byte
          CALL   #ack             ; wait for ACK
          MOV.B  22(SP), R8       ; data to write –high byte
          CALL   #sbyte           ; send high data byte
          CALL   #ack             ; wait for ACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; read byte protocol   6
; ==========================
```

```
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     data          in
;                     NACK          out
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   low data byte
;   unused
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #008h         ; push the command for battery temperature
;        PUSH   #00h          ; reserve a byte for data (unused)
;        PUSH   #00h          ; reserve a byte for data
;        CALL   #rbp
;        POP    R10           ; pop data to R10
;        ADD    #06h, SP      ; free the address and command space
;        POP    R9        ; pop the error code
;        CMP    #00h, R9
;        JZ     error         ;  unsuccessful – run error routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rbp
          PUSH   SR
          PUSH   R7
          PUSH   R8
```

```
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send a 1 for read
          CALL   #ack             ; wait for ACK
          CALL   #rbyte           ; receive data byte
          CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  R14, 22(SP)      ; copy the data byte to the stack
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; read word protocol  7
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
```

```
;                       acknowledge   in
;                       start         out
;                       address       out
;                       R/W           out
;                       acknowledge   in
;                       data          in
;                       acknowledge   in
;                       data          in
;                       acknowledge   in
;                       stop          out
;
; Stack parameters
; ----------------
;   error_flag          low mem
;   low data byte
;   high data byte
;   command
;   address             high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #008h         ; push the command for battery temperature
;        PUSH   #00h          ; reserve a byte for data
;        PUSH   #00h          ; reserve a byte for data
;        CALL   #rwp
;        POP    R10           ; pop data to R10
;        POP    R11           ; pop data to R11
;        ADD    #04h, SP      ; free the address and command space
;        POP    R9        ; pop the error code
;        CMP    #00h, R9
;        JZ     error     ;  unsuccessful – run error routine
;
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rwp
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
```

```
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send a 1 for read
          CALL   #ack             ; wait for ACK
          CALL   #rbyte           ; receive low data byte
          MOV.B  R14, 22(SP)      ; store low data byte on stack
          CALL   #sack            ; send an  ACK
          CALL   #rbyte           ; receive high data byte
          CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  R14, 24(SP)      ; store high data byte on stack
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; block write protocol  8
; ========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack, memory block
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     R/W           out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     byte count    out
;                     acknowledge   in
```

```
;                     data          out
;                     acknowledge   in
;                      :
;                     repeat  data, acknowledge n times
;                      :
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   byte count
;   block pointer
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #0020h        ; push the command for battery temperature
;        PUSH   #00550h       ; pointer to begining of block
;        PUSH   #04h          ; reserve a byte for byte count
;        CALL   #blkw
;        POP    R10           ; pop count to R10
;        ADD    #06h, SP
;        POP    R9        ; pop the error code
;        CMP    #00h, R9
;        JZ     error         ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
blkw
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          MOV    24(SP), R9       ; block start address
          MOV.B  #00h, R15        ; counter to loop through bytes
          CALL   #sbit            ; send start and address
          CALL   #sendzero        ; send a 0 for write
```

```
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          MOV.B  22(SP), R8       ; copy the byte count
          CALL   #sbyte           ; send the byte count
          CALL   #ack             ; wait for ACK
blk_rep   MOV.B  0(R9), R8        ; move the data from the block to data
          CALL   #sbyte           ; send data byte
          CALL   #ack             ; wait for ACK
          INC    R15              ; increment block position counter
          INC    R9               ; increment the block pointer
          CMP    22(SP), R15      ; check for end of data
          JNZ    blk_rep          ; repeat until end
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; block read protocol  9
; ==========================
; Registers affected: none
; Memory affected   : maximum of 40 bytes on stack, memory block
; --------------------------
; Description         Step        Direction
;                     -------------------------
;                     start         out
;                     address       out
;                     W             out
;                     acknowledge   in
;                     command       out
;                     acknowledge   in
;                     start         out
;                     address       out
;                     R             out
;                     acknowledge   in
;                     byte count    in
;                     acknowledge   out
```

```
;                     data          in
;                     acknowledge   out
;                      :
;                     repeat  data, acknowledge n times
;                      :
;                     nack          out
;                     stop          out
;
; Stack parameters
; -----------------
;   error_flag        low mem
;   byte count
;   block pointer
;   command
;   address           high mem
;
; Sample usage
; --------------
;        PUSH   #00h          ; error code space
;        PUSH   #000Bh        ; push the address onto the stack
;        PUSH   #0020h        ; push the command for battery temperature
;        PUSH   #00550h       ; pointer to begining of block
;        PUSH   #00h          ; reserve a byte for byte count
;        CALL   #blkr
;        POP    R10           ; pop count to R10
;        ADD    #06h, SP
;        POP    R9            ; pop the error code
;        CMP    #00h, R9
;        JZ     error         ;  unsuccessful – run error routine
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
blkr
          PUSH   SR
          PUSH   R7
          PUSH   R8
          PUSH   R9
          PUSH   R14
          PUSH   R15
          PUSH   R10
          PUSH   R11
          PUSH   R12
          PUSH   R13
          MOV.B  #00h, R15        ; counter to loop through bytes
          MOV.B  26(SP), R8       ; command parameter
          MOV.B  28(SP), R7       ; address parameter
          MOV    24(SP), R9       ; block start address
          CALL   #sbit            ; send start and address
```

```
          CALL   #sendzero        ; send a 0 for write
          CALL   #ack             ; wait for ACK
          CALL   #sbyte           ; send command code
          CALL   #ack             ; wait for ACK
          CALL   #rsbit           ; send repeated start and address
          CALL   #sendone         ; send 1 for read
          CALL   #ack             ; wait for ack
          CALL   #rbyte
          MOV.B  R14, 22(SP)      ; push the count onto the stack
          CALL   #sack            ; send an ack
rblk_rep  CALL   #rbyte           ; receive data byte
          MOV.B  R14, 0(R9)       ; move the data from the block to data
          INC    R15              ; increment block position counter
          INC    R9               ; increment the block pointer
          CMP    22(SP), R15      ; check for end of data
          JZ     blk_done         ; repeat until end
          CALL   #sack            ; send an ACK
          CMP    22(SP), R15      ; check for end of data
          JNZ    rblk_rep         ; repeat until end
blk_done  CALL   #nack            ; send a NACK
          CALL   #pbit            ; send stop
          MOV.B  #01h, 30(SP)     ; return a 1 for successful transfer
          POP    R13
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

;***********************************************************************
; low level subroutines, implement common parts of all transfers
;***********************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send start condition and an address
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbit    ; release both lines
          BIC.B   #DNC, DIR
        ; check for free bus
          MOV     #05h, R10     ; counter for ticking off 50 micro sec
wait
          MOV.B   IN, R11       ; copy the inputs to R11
```

```
          AND     #DNC, R11     ; mask all but inputs
          CMP     #DNC, R11     ; if either SDA or SCL is low the bus is busy
          JNZ     busy
          DEC     R10           ; decrement the counter
          JNZ     wait
rsbit     BIC.B   #DNC, DIR     ; redundant line release so that a repeated
                                ;start can use the same routine
        ; send out the start condition
          BIS.B   #SDA, DIR
          BIS.B   #SCL, DIR
        ; send out the address
          MOV.B   #07h, R13     ; counter for the 7 bit address
          MOV.B   R7, R11       ; copy the address to R11
ashift    RLA.B   R11           ; rotate left so MSB of 7 bit address is in position 7
          MOV.B   R11, R12      ; copy so it can be masked without loss of data
          AND.B   #080h, R12    ; mask all but MSB
          CMP.B   #00h, R12     ; compare it to 0
          JNZ     one
          CALL    #sendzero     ; send a one
          JMP     zero
one       CALL    #sendone      ; send a zero
zero      DEC     R13           ; decrement the 7 counter
          JNZ     ashift        ; if <7 bits have been sent repeat
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; No ACK received case, do a STOP then busy
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbusy     CALL    #pbit          ; generate a stop, then  busy code
          ADD    #02h, SP        ; remove stuff from stack since RET wasn't used
          MOV.B  #00h, 28(SP)    ; error code
          POP    R13             ; restore registers
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Handle a busy system
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
busy3     ADD     #06h, SP      ; remove extra data from sendone or sendzero
busy      ADD     #02h, SP      ; remove data from stack since RET wasn't used
```

```
          MOV.B  #00h, 28(SP)   ; error code
          POP    R13            ; restore registers
          POP    R12
          POP    R11
          POP    R10
          POP    R15
          POP    R14
          POP    R9
          POP    R8
          POP    R7
          POP    SR
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a 1
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sendone   PUSH    R10
          PUSH    R11
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          MOV.B   #04h, R11      ; adjust number of loops for correct time
clkext    MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy3          ; took too long for it to rise--arbitration
          AND.B   #SCL, R10      ; check SCL
          JZ      clkext
       ; arbitration:  make sure that SDA is really high, if not arbitration was
       ;lost to another master on the bus
          MOV.B   IN, R10        ; check the bus
          AND.B   #SDA, R10      ; mask all but SDA
          JZ      busy3
          BIS.B   #SCL, DIR      ; release SCL
          MOV.B   IN, R10        ; detect a repeated start condition
          AND.B   #SDA, R10
          JZ      busy3
          POP     R11
          POP     R10
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a 0
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sendzero
          PUSH    R10
          PUSH    R11
```

```
          BIS.B   #SDA, DIR      ; put a 0 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          MOV     #04h, R11      ; counter to time the rise of SCL
clke      MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy3          ; arbitration-- took too long for SCL to rise
          AND.B   #SCL, R10      ; check SCL
          JZ      clke
          BIS.B   #SCL, DIR      ; release SCL
          POP     R11
          POP     R10
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Wait for acknowledge
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
ack
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
          ;MOV     #008h, R11    ; counter for approx. 50 micro sec
          MOV     #02FFh, R11    ; counter extended PS100Z-200 clock low extends here
clkex     MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy           ;arbitration--took too long to rise
          AND.B   #SCL, R10      ; check SCL
          JZ      clkex
        ; wait for the ack signal (SDA going low)
          MOV     #05F6h, R11    ; counter to timeout for a NACK
wack
          MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      sbusy          ; timeout NACK-- send P bit and retry
          AND.B   #SDA, R10      ; mask all but SDA
          JNZ     wack           ; either see the ACK or wait here for timeout
          BIS.B   #SCL, DIR      ; put 0 on SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Transmit stop condition
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
pbit
          BIS.B   #SDA, DIR      ; put a 0 on SDA
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          NOP
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          NOP
```

```
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Send a byte (uses sendzero and sendone)
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sbyte
        ; send out the data
          MOV.B   #08h, R13      ; counter for approx. 50 micro sec.
          MOV.B   R8, R11        ; copy the data to R11
dshift
          MOV.B   R11, R12
          AND.B   #080h, R12     ; mask all but MSB
          CMP.B   #00h, R12      ; compare it to 0
          JNZ     on
          CALL    #sendzero      ; send a one
          JMP     zer
on        CALL    #sendone       ; send a zero
zer
          RLA.B   R11            ; rotate left so data to go out is in MSB
          DEC     R13            ; decrement the 8 counter
          JNZ     dshift
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; receive a byte
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rbyte
          MOV.B   #08h,  R12     ; counter for 8 data bits
          BIC.B   #SDA, DIR      ; put a 1 on SDA
rrep
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          MOV     #0035h, R13    ; counter extended PS100Z–200 clock low extends here
clkxtn    MOV.B   IN, R10        ; check the bus
          DEC     R13
          JZ      busy           ; if it took too long to rise abort
          AND.B   #SCL, R10
          JZ      clkxtn         ; if SCL is not high wait
          MOV.B   IN, R10        ; check the bus
          RLA.B   R11            ; move over for input as LSB
          AND.B   #SDA, R10      ; check SDA
          JZ      inzero
          BIS.B   #01h, R11      ; set the LSB
inzero    DEC     R12
          JNZ     rrep           ; repeat for the rest of the byte
          MOV.B   R11, R14       ; copy the data to the variable
          BIS.B   #SCL, DIR      ; put a 0 on SCL
```

```
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send a NACK
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
nack
          BIC.B   #SDA, DIR      ; put a 1 on SDA
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          BIC.B   #SCL, DIR      ; put a 1 on SCL
        ; due to clock low extending, don't outrun the slave
        ; wait for the SCL line to rise
clkn      MOV.B   IN, R10
          AND.B   #SCL, R10      ; check SCL
          JZ      clkn
          BIS.B   #SCL, DIR      ; release SCL
          RET
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; send a ACK
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sack      ; wait for SDA release by slave
          MOV     #000Fh, R11    ; counter for approx. 50 micro sec
swaita    MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy
          AND.B   #SDA, R10      ; check SCL
          JZ      swaita
          BIS.B   #SDA, DIR      ; pull SDA low
          BIC.B   #SCL, DIR      ; put a 1 on SCL
          MOV     #000Fh, R11    ; counter for approx. 50 micro sec
swait     MOV.B   IN, R10        ; check the bus
          DEC     R11
          JZ      busy
          AND.B   #SCL, R10      ; check SCL
          JZ      swait
          BIS.B   #SCL, DIR      ; put a 0 on SCL
          BIC.B   #SDA, DIR      ; release SDA
          RET
;--- clear LCD
show_clr
          MOV    #15, r5         ; clear display memory
show_clr1
          MOV.b  #0, LCD1-1(r5)
          DEC    r5
          JNZ    show_clr1
          RET

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
; LCD  Definitions
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
LCD_TYPE
;–––STK/EVK LCD
a       .equ     01h
b       .equ     02h
c       .equ     10h
d       .equ     04h
e       .equ     80h
f       .equ     20h
g       .equ     08h
h       .equ     40h
;––– character definitions

LCD_Tab .byte  a+b+c+d+e+f                ; displays ”0”
        .byte   b+c                       ; displays ”1”
        .byte   a+b+d+e+g                 ; displays ”2”
        .byte   a+b+c+d+g                 ; displays ”3”
        .byte   b+c+f+g                   ; displays ”4”
        .byte   a+c+d+f+g                 ; displays ”5”
        .byte   a+c+d+e+f+g               ; displays ”6”
        .byte   a+b+c                     ; displays ”7”
        .byte   a+b+c+d+e+f+g             ; displays ”8”
        .byte   a+b+c+d+f+g               ; displays ”9”
        .byte   0                         ; displays ”:” blank
        .byte   g                         ; displays ”;” –
        .byte   a+d+e+f                   ; displays ”<” [
        .byte   d+g                       ; displays ”=”
        .byte   a+b+c+d                   ; displays ”>” ]
        .byte   a+b+e+g                   ; displays ”?”
        .byte   a+b+d+e+f+g               ; displays ”@”
        .byte   a+b+c+e+f+g               ; displays ”A”
        .byte   c+d+e+f+g                 ; displays ”B” b
        .byte   a+d+e+f                   ; displays ”C”
        .byte   b+c+d+e+g                 ; displays ”D” d
        .byte   a+d+e+f+g                 ; displays ”E”
        .byte   a+e+f+g                   ; displays ”F”
        .byte   a+b+c+d+f+g               ; displays ”G”
        .byte   b+c+e+f+g                 ; displays ”H”
        .byte   b+c                       ; displays ”I”
        .byte   b+c+d+e                   ; displays ”J”
        .byte   0                         ; displays ”K”
        .byte   d+e+f                     ; displays ”L”
        .byte   a+b+c+e+f                 ; displays ”M”
        .byte   c+e+g                     ; displays ”N” n
        .byte   c+d+e+g                   ; displays ”O” o
```

```
        .byte   a+b+e+f+g               ; displays ”P”
        .byte   0                       ; displays ”Q”
        .byte   e+g                     ; displays ”R” r
        .byte   a+c+d+f+g               ; displays ”S”
        .byte   d+e+f+g                 ; displays ”T” t
        .byte   c+d+e                   ; displays ”U” u
        .byte   0                       ; displays ”V”
        .byte   0                       ; displays ”W”
        .byte   0                       ; displays ”X”
        .byte   b+c+d+f+g               ; displays ”Y”
        .byte   a+b+d+e+g               ; displays ”Z” 2
LCD_Tab_End
        .even
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; Interrupt vectors
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          ; .sect     ”Int_Vect”,0FFFFh–1   ; without monitor
          ; .word     RESET                 ; POR, ext. Reset, Watchdog
          ; .end
           .sect     ”Int_Vect”,05FFh–1    ; with monitor
           .word     RESET                 ; POR, ext. Reset, Watchdog
           .end
```

Appendix D SMBus Slave Software

```
;*******************************************************************
; SMBus slave program
;*******************************************************************
USER_END  .set   0FFFFh
Device    .set   325
;--- Control register definitions
SRE             .equ    0h
IE1             .equ    0h
IE2             .equ    01h
IFG1            .equ    02h
IFG2            .equ    03h
WDTCTL          .equ    0120h
WDTHold         .equ    80h
WDT_wrkey       .equ    05A00h
CPUOFF          .set    10h
OSCOFF          .set    20h
;PORT 0

P0IE        .equ    015h
P0DIR       .equ    012h
P0IN        .equ    010h
P0OUT       .equ    011h
P0IES           .equ    014h
P0IFG           .equ    013h

; port 2
P2IN        .equ    028h
P2OUT       .equ    029h
P2IE            .equ    02Dh
P2SEL           .equ    02Eh
P2DIR           .equ    02Ah
LCD1            .equ    031h
LCDM            .equ    030h
OUT             .equ    P0OUT
IN              .equ    P0IN
DIR             .equ    P0DIR

SDA     .equ    010h
SCL             .equ    020h
DNC             .equ    030h
; table location to hold the scroll text
Table           .equ    0300h
;*******************************************************************
; Reset : Initialize processor
;*******************************************************************
```

```
         .sect ”onreset”,0FFFEh
          .word   INIT
          .sect ”intv”, 0FFE0h
          .word   rbp
          .sect ”MAIN”,0C000h
INIT
        .if   Device = 325
          MOV    #03C0h, SP
        .endif
        .if   Device = 337
          MOV    #05D0h, SP
        .endif
          MOV    #(WDTHold+WDT_wrkey),&WDTCTL ; Stop Watchdog Timer
;set up port2
          BIC.B  #DNC, DIR     ; bits 1 and 0 are in
          BIC.B  #DNC, OUT     ; set SDA and SCL high
          BIS.B  #SDA, P0IE    ; enable interrupt on P0.4
          BIS.B  #SDA, P0IES   ; direction H/L on P0.4
RESET
nready ; bus not idle / no start condition
waddr  ; wrong device address returns here

          MOV.B  #000h, error   ; reset the error flag
          MOV.B  #055h, address ; set the address as 55h
          MOV.B  #09h, data     ; set the data as 09h
          EINT                  ; general interrupt enable

;***********************************************************************
; User program begining
;***********************************************************************
repeat
          JMP    repeat        ; wait for interrupt

;***********************************************************************
; User program end
;***********************************************************************
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; receive byte code   (interrupt goes to here)
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
rbp
          ;PUSH  R11            ; if the start condition is known to be slow, can add these
          ;PUSH  R12
          ;PUSH  R13
          ;PUSH  R14
          BIC.B  #SDA, P0IFG    ; clear the flag by writing a 0 to it
```

```
;*******************************************
; check for a start condition
;*******************************************
wait2     MOV.B  IN, R11       ; copy the inputs to R11
          DEC    R9
          JZ     done
          AND    #DNC, R11     ; mask all but inputs
          CMP    #SCL, R11     ; see if SDA L and SCL H
          JNZ    wait2         ; if not wait
wait3     MOV.B  IN, R11       ; copy the inputs to R11
          DEC    R9
          JZ     done
          AND    #DNC, R11     ; mask all but inputs
          JNZ    wait3         ; if not wait
;*******************************************
; start condition achieved
;*******************************************
          MOV   #007h, R13     ; counter for the 7 addr bits
          MOV   #0000h, R12    ; register to hold the address as shifted in

taddr     MOV.B  IN, R14       ; make sure SCL is low
          DEC    R9
          JZ     done
          AND    #SCL, R14
          JNZ    taddr
addr      MOV.B  IN, R14       ; read the port
          DEC    R9
          JZ     done
          MOV.B  R14, R11      ; copy the data
          AND    #SCL, R14     ; mask all but SCL
          JZ     addr          ; wait for SCL to rise
          AND    #SDA, R11     ; mask all but SDA
          JZ     szero         ; if 0 just shift
          RLA.B  R12           ; left shift
          BIS.B  #0001h, R12   ; if 1 set LSB and shift
          JMP    sone
szero
          RLA.B  R12           ; arithmetic left shift   <-0
sone      DEC    R13           ; decrement the counter
          JNZ    taddr         ; if less than 7 bits repeat
          MOV.B  address, R13  ; copy the device address to R13
          CMP.B  R13, R12      ; compare it to the received address
          JNZ    waddr         ; wrong address, deal with it appropriately
;*******************************************
; this device so keep going
;*******************************************
```

```
          ; read the R/W bit
rwait2    MOV.B  IN, R11    ; wait for the SCL to drop
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JNZ     rwait2
rwait     MOV.B  IN, R11    ; wait for the SCL to rise
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JZ     rwait
          MOV.B  IN, R11    ; copy the port data
          AND.B  #SDA, R11  ; mask all but SDA
          JNZ    done       ; 0 for write jmp if a 1 (read)
;*************************************************************
; send the ACK bit
;*************************************************************
rwait3    MOV.B  IN, R11    ; wait for the SCL to drop
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JNZ    rwait3
          BIS.B  #SDA, DIR  ; drive SDA low
rwait4    MOV.B  IN, R11    ; wait for the SCL to rise
          DEC    R9
          JZ     done
          AND.B  #SCL, R11   ; mask all but SCL
          JZ     rwait4
rwait5    DEC    R9
          JZ     done
          MOV.B  IN, R11     ; wait for the SCL to drop
          AND.B  #SCL, R11   ; mask all but SCL
          JNZ    rwait5
          BIC.B  #SDA, DIR   ; release SDA
;***********************************************
; ready to receive a byte
;***********************************************

          MOV    #008h, R13   ; counter for the 8 data bits
          MOV    #0000h, R12  ; register to hold the address as shifted in
tdat      MOV.B  IN, R14      ; make sure SCL is low
          DEC    R9
          JZ     done
          AND    #SCL, R14
          JNZ    tdat
dat       MOV.B  IN, R14     ; read the port
```

```
          MOV.B  R14, R11    ; copy the data
          DEC    R9
          JZ     done
          AND    #SCL, R14   ; mask all but SCL
          JZ     dat         ; wait for SCL to rise
          AND    #SDA, R11   ; mask all but SDA
          JZ     szeroc      ; if 0 just shift
          RLA.B  R12
          BIS.B  #0001h, R12 ; if 1 set LSB and shift
          JMP    szerob
szeroc    RLA.B  R12         ; arithmetic left shift
szerob
          DEC    R13         ; decrement the counter
          JNZ    tdat
          MOV.B  R12, data   ; copy to RAM

;***************************************************************
; send the ACK bit
;***************************************************************
rwait6    MOV.B  IN, R11    ; wait for the SCL to drop
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JNZ    rwait6
          BIS.B  #SDA, DIR  ; drive SDA low
rwait7    MOV.B  IN, R11    ; wait for the SCL to rise
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JZ     rwait7
rwait8    MOV.B  IN, R11    ; wait for the SCL to drop
          DEC    R9
          JZ     done
          AND.B  #SCL, R11  ; mask all but SCL
          JNZ    rwait8
          BIC.B  #SDA, DIR  ; release SDA

done
          ;POP R14          ; if start condition is know to be slow, can add these
          ;POP R13
          ;POP R12
          ;POP R11
          MOV   #0FFh, R9
RETI
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; variables
```

```
; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
address    .equ       0248h  ; device address
data       .equ       0242h  ; data to be sent
datin      .equ       0244h  ; data received
error      .equ       0246h  ; error flag
```

Appendix E Contents of ASCII.txt

```
@B500
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
B7 12 8F 1F 3A 3D BD 13 BF 3F 00 00 00 00 00 00
00 BB BC A5 9E AD A9 3F BA 12 96 00 A4 B3 98 9C
AB 00 88 3D AC 94 00 00 00 3E 8F 00 00 00 00 00
00 BB BC A5 9E AD A9 3F BA 12 96 00 A4 B3 98 9C
AB 00 88 3D AC 94 00 00 00 3E 8F 00
q
```

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.