AN1292

Adding a Voice User Interface to M68HC05 Applications

By Derrick B. Forte and Hai T. Nguyen
CSIC Development Tools
Austin, Texas

Introduction

As embedded microcontroller-based products become more sophisticated, additional emphasis is being placed on the design and implementation of their user interfaces. Visually based interfaces are commonly implemented with LCDs, LEDs, fluorescent displays, and lights. Many of these components can be controlled directly by an application's processor without using additional components. Voice-based user interfaces, on the other hand, are often implemented with speech synthesizers, speech processors, sound generators, and digital signal processors which operate in conjunction with an application's main processor. In addition to a processor dedicated to the generation of speech, designs frequently require a memory device to hold the data used by the processor, a loudspeaker, and audio amplification circuitry. The added cost for components and space has limited the implementation of speech-based user interfaces to higher end products and products for the visually impaired. This application note discusses adding a voice-based user interface to an application based on the Freescale MC68HC(7)05J1A microcontroller. In particular, interfacing members of the M68HC05 MCU Family to the Information Storage Devices (ISD) 1000 and 2500 family of voice record/playback devices is highlighted. The development of an audible thermometer application concludes the discussion.

Design Alternatives

Most applications that use speech as part or all of their user interface utilize any one of a number of speech processors and synthesizers on the market. The speech output by these devices usually is stored as noncompressed or compressed digital data in on-chip or external memory. In devices designed specifically for high-volume applications, speech data usually is stored in on-chip ROM.

However, most speech processors and synthesizers designed for the general market require more flexibility than can be offered by using on-chip ROM. Data for these processors is provided by a programmed external memory device. Depending on whether audio amplification is done by the speech processor or not, external circuitry may be needed to interface the processor to a loudspeaker. A block diagram of a typical speech system is illustrated in **Figure 1**.

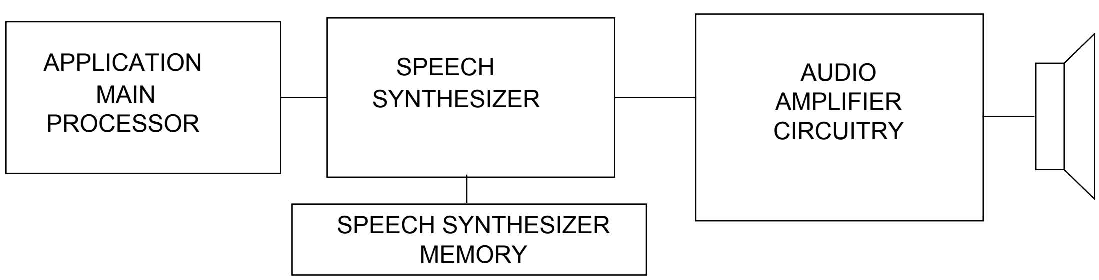


Figure 1. Typical Speech System Design

The ISD 1000 and 2500 series of voice record/playback devices discussed in this application note integrate all the circuitry needed to record and play back audio signals on a single device. The speech processor, memory, and audio amplifier functional blocks needed to implement a speech interface are all integrated on this device. The members of the device family differ in the length of their recording times, ranging from 10 seconds to 90 seconds. The devices are designed to operate in a number of standalone recording and playback modes or under the control of an external microcontroller. The ease with which these devices can be interfaced with microcontrollers makes them ideal for adding a voice-based user interface to an application based on a member of Motorola's M68HC05 Family of MCUs. The remainder of this note discusses using one of these devices to add a voice-based user interface to a simple M68HC05 MCU-based application, namely a digital thermometer.

Audible Thermometer Feature Definition

The system design of the audible thermometer begins with the definition of the application's feature set. The audible thermometer senses ambient temperature and outputs the temperature reading in a pre-recorded human voice. The thermometer is capable of sensing temperatures from –55 to +125 degrees Celsius in 0.5-degree increments. The thermometer powers up and remains in a low-power idle state until the user presses a button. Pushing the button wakes up the thermometer, causing it to acquire and output a temperature reading. After completing these tasks, the system returns to a low-power idle state.

Audible Thermometer Hardware Design

The system design of digital thermometers is a well-established paradigm in the design of embedded systems applications. The audible thermometer follows this model and its hardware design can be divided into two main functional blocks:

1. Temperature acquisition and conversion – Senses ambient temperature and converts the reading to the digital domain
2. Audio processing and output – Outputs the temperature reading in a human voice

To illustrate the ease with which a voice interface can be added to a Freescale M68HC05 MCU-based application, the Freescale MC68HC(7)05J1A microcontroller was chosen as the main system processor for this application. This device is the simplest and the most inexpensive member of Motorola's M68HC05 Family of microcontrollers. The MC68HC(7)05J1A's main on-chip peripherals include an 8-bit free-running timer and 14 bidirectional I/O pins. The MC68HC(7)05J1A's simplicity constrains the role that it plays in the hardware implementation of these two blocks.

The temperature acquisition and conversion block consists of circuitry that senses the application's ambient temperature and converts it to a suitable electrical signal for processing by the system's microcontroller. This block typically consists of a temperature sensor, signal conditioning circuitry, and an A/D converter. The temperature sensor is capable of varying a voltage or current signal in proportion to its ambient temperature. The signal is then processed by some form of analog conditioning circuitry. The conditioning circuitry design is heavily dependent on the accuracy, sensitivity, and noise rejection parameters of the application's specifications and its components. This circuitry may amplify, filter, and linearize the signal in preparation for its conversion to the digital domain by the A/D converter. Once in digital form, the signal can be processed by the microcontroller. In most M68HC05-based applications, the temperature sensor and conditioning circuitry generate and process an analog signal for use by the MCU's on-chip A/D converter peripheral. However, since the MC68HC(7)05J1A does not have an on-chip A/D converter, an external A/D converter is needed to implement this block completely. The added cost and space required by an external A/D converter led to the selection of the Dallas Semiconductor DS1820 One-Wire Digital Thermometer to implement the temperature acquisition and conversion block in this application. The DS1820 is a 3-pin device that integrates the temperature sensor, conditioning circuitry, and A/D converter needed to implement this block on a single device. In addition, the DS1820 also has nine bytes of scratchpad RAM and two bytes of EEPROM memory. Using this device results in substantial cost and space savings. The temperature sensed by this device is available to the microcontroller as a 9-bit binary number which can be read serially from a single pin.

The audio processing and output block in this application serves to output the temperature read in a human voice. As mentioned earlier, the ISD 1000 and 2500 series voice record/playback devices contain most of the circuitry needed to implement this block. The high degree of integration provided by this device allows this block to be implemented using this device, a few passive components, and a loudspeaker. The device selected for use in the audible thermometer is the ISD2560. This device is capable of recording and playing back 60 seconds of sound and/or speech. The ISD2560 records by sampling a speech or sound signal at 8 kHz and storing the samples as discrete analog levels in storage cells. The ISD2560 has 480 K of such cells mapped in a memory space that is divided into 600 addresses. Sound recording can be initiated at any one of the 600 addresses and is stopped either by the manipulation of device control signals or by reaching the end of the device's memory space. To separate recordings, special end of message (EOM) markers are placed in memory at the end of each recording. This gives the ISD2560 the ability to record a number of separate recordings or messages and play them back as many times or in any sequence desired. The audible thermometer uses this feature of the ISD2560 to output a sequence of pre-recorded phrases that correspond to the temperature read by the DS1820. In the thermometer, the ISD2560 is pre-recorded with phrases for the numbers 0 through 19, the numbers 20 through 90 in increments of 10, and the words "one hundred," "point," "degrees," "negative," and "Celsius." (See the *Design Manual for ISD1000A Family* for recording instructions.) These phrases are recorded at addresses in the ISD2560's memory space that are 16 units apart starting at address $0000. This allots a time of 1.5 seconds per phrase. The ISD2560 signals encountering an EOM marker by pulsing the /EOM pin low and then high. The signal can be used by an external controller to concatenate a sequence of messages.

Although the ISD2560 is capable of operating in a number of standalone or operational modes, the MC68HC(7)05J1A interfaces with the device at its microcontroller interface.

The following describes the ISD2560's microcontroller interface pins and their functions:

1. A0–A9 – Address lines 0–9: Inputs used to access the 600 addresses within the device's memory space. Although the number of lines allows the selection of 1024 addresses, only addresses 00 to 257 hex are valid.
2. /CE – Chip Enable: An active low pin that enables recording and playback operations
3. PD – Powerdown: An active high pin that puts the device in a low-power idle state.
4. P/R – Playback/ Record: A pin that enables device recording when it is high and enables playback operations when it is low.
5. /EOM – End of Message: An active low pin that pulses for 12.5 msec after the end of a message.
6. /OVF – Overflow: An active low pin that signals the end of the device's memory space. This signal can be used to cascade more than one ISD device together for greater message storage capacity.

After defining the system's hardware functional blocks of the audible thermometer and selecting the components that comprise the blocks, the system block diagram in **Figure 2** was derived for the audible thermometer.

Schematics for the application's hardware design are located in **Audible Thermometer Schematics**.

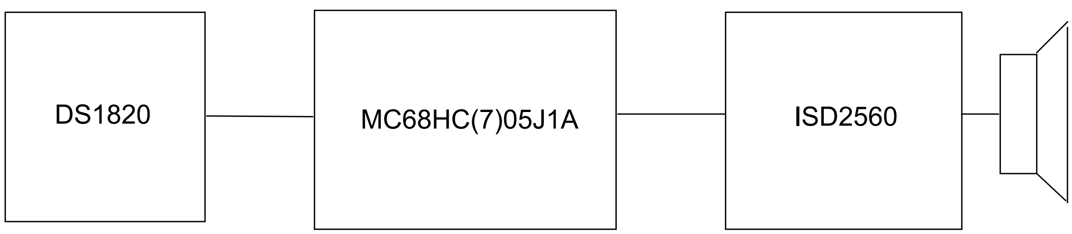


Figure 2. Audible Thermometer System Block Diagram

Audible Thermometer Software Design

The audible thermometer's system software can be divided into the main program functions and the low-level functions that interface the MC68HC(7)05J1A to the DS1820 and the ISD2560. The low-level driver routines are discussed first, since the main program routines are built on them.

When given the proper command sequence, the Dallas Semiconductor DS1820 One-Wire Digital Thermometer is designed to acquire a temperature measurement within one second and convert it to a 9-bit digital word. The temperature measured is mapped into a range of 9-bit words that span from –55 to +125 degrees Celsius in 0.5-degree increments. The upper byte of a word indicates whether the temperature read is above or below 0 degrees Celsius. An upper byte value of $FF corresponds to a negative temperature and a value of $00 corresponds to a positive temperature. The lower byte values range from $01 to $FA for positive temperatures and from $FF to $92 for negative ones. When a temperature is read, the converted word is stored, least significant byte first, in the first two bytes of the DS1820's scratchpad RAM memory. The device interfaces with a microcontroller over a single serial line using a half-duplex serial protocol. The protocol prescribes that the MCU initiate and sustain all communications with the DS1820. This protocol supports a full-featured command set that provides the microcontroller with complete control over the DS1820's operation. The DS1820 command set includes commands to read and write scratchpad RAM memory, to read and write EEPROM memory, and to perform a temperature reading and conversion operation. Although the DS1820 is a multi-featured device, the audible thermometer only uses the commands required to perform a temperature reading and conversion operation and read the 9-bit data word from the DS1820. In this application, the DS1820 interfaces to the MC68HC(7)05J1A at its PB5 bidirectional input/output (I/O) pin. Since the DS1820's protocol is not a standard, the MC68HC(7)05J1A must manipulate or "bit bang" the PB5 pin to communicate with a DS1820.

The DS1820's serial protocol supports three communication functions: reset, read, and write.

A reset sequence initializes a DS1820 and prepares it to receive a command from the MCU. A DS1820 reset can be initiated only by the microcontroller and consists of a reset pulse from the microcontroller followed by an acknowledgment pulse from the DS1820. This requires that after driving the serial line to output the reset pulse, the MCU's I/O pin must be changed from an output to an input to receive the acknowledgment pulse. Since setting the I/O line as an input three-states the serial line, a pullup resistor is needed to pull the serial line high while the microcontroller is not driving it. If an acknowledgment pulse is not received from the DS1820 within 15 to 60 microseconds from the rising edge of the reset pulse, the DS1820 is considered to be inoperative. **Figure 3** illustrates the timing requirements for a DS1820 reset operation.

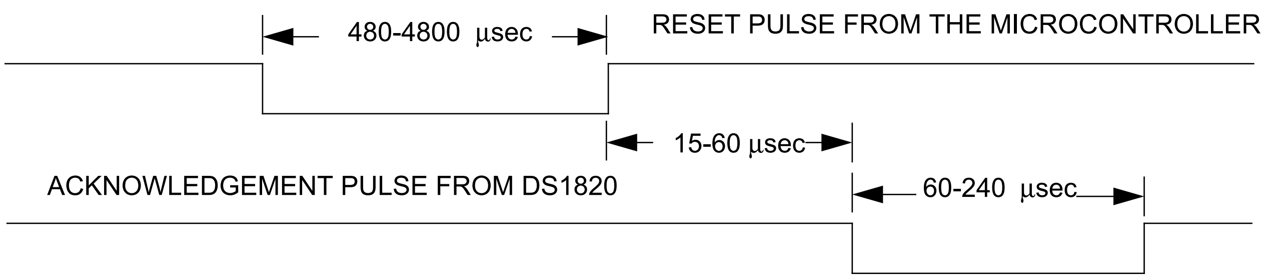


Figure 3. DS1820 Reset Sequence

The MC68HC(7)05J1A sends commands and data to the DS1820 using the device's write protocol. The microcontroller initiates a write cycle or time slot by pulling the serial line low. A write cycle must be a minimum of 60 microseconds long with a minimum recovery time of 1 microsecond between cycles. Data is output least significant bit first with each bit requiring one complete write cycle. **Figure 4** illustrates the timing requirements for writing a 1 or 0 to the DS1820.

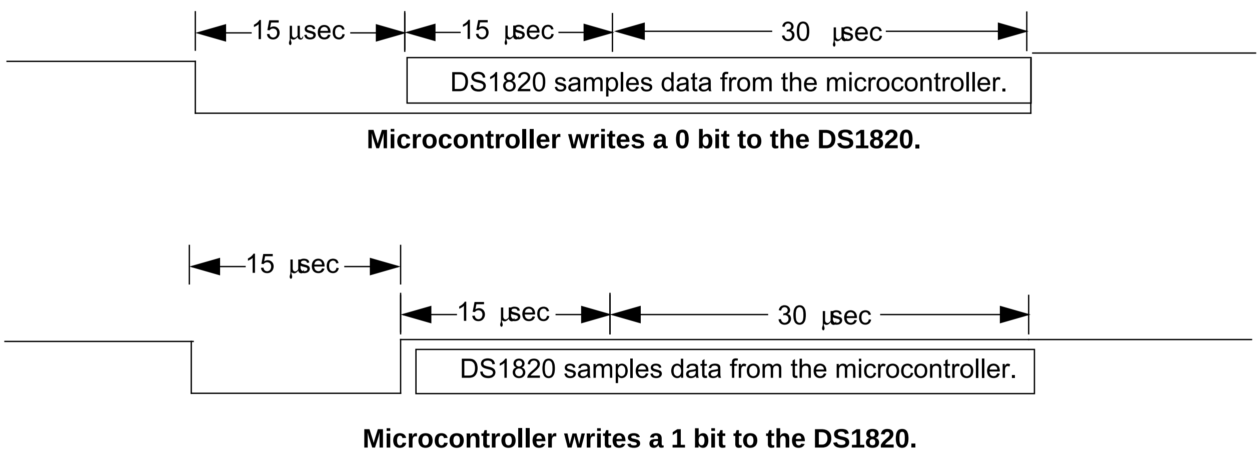
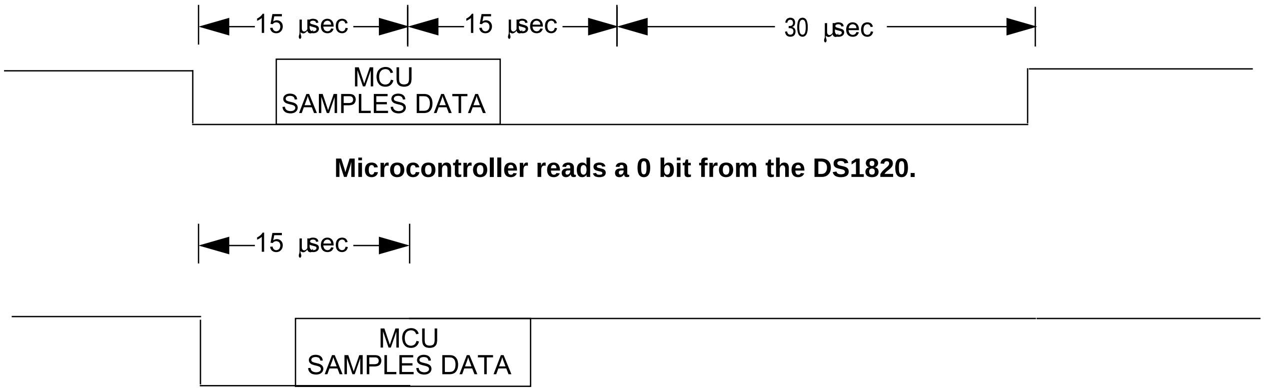


Figure 4. Microcontroller to DS1820 Write Cycle

The MC68HC(7)05J1A reads data from the DS1820 using the device's read protocol. The microcontroller initiates a read cycle or time slot by pulling the serial line low for a minimum of one microsecond. The DS1820 outputs a valid bit 15 microseconds after the start of the read cycle. Therefore, the MCU must change the I/O line driving the serial line from an output to an input before the DS1820 starts to output data. The pullup resistor on the serial line pulls up the line until the DS1820 is ready to output a bit. A read cycle must be a minimum of 60 microseconds with minimum recovery time of 1 microsecond between cycles. The DS1820 outputs data least significant bit first with each bit requiring one full read cycle. **Figure 5** illustrates the timing requirements for reading a 1or 0 from the DS1820.



Microcontroller reads a 0 bit from the DS1820.

Microcontroller reads a 1 bit from the DS1820.

Figure 5. DS1820 Read Cycle

The ISD2560 driver functions enable the device to play back a sequence of pre-recorded phrases under the direction of the MC68HC(7)05J1A. The MC68HC(7)05J1A performs this simple sequence of I/O port operations to cause the ISD2560 to output a single pre-recorded phrase:

1. Pulls the ISD2560's PD low, taking the device out of powerdown mode
2. Sets the ISD2560's P/R pin high, enabling playback operation
3. Places the starting address of the message on the ISD2560's address bus
4. Pulses the ISD2560's /CE pin low then high for a minimum of 100 nanoseconds
5. Waits for a falling edge on ISD2560's /EOM pin, indicating that an EOM marker has been encountered
6. Waits for the rising edge on the ISD2560's /EOM pin, indicating the end of the EOM pulse

Figure 6 illustrates a timing diagram for the ISD2560's signals.

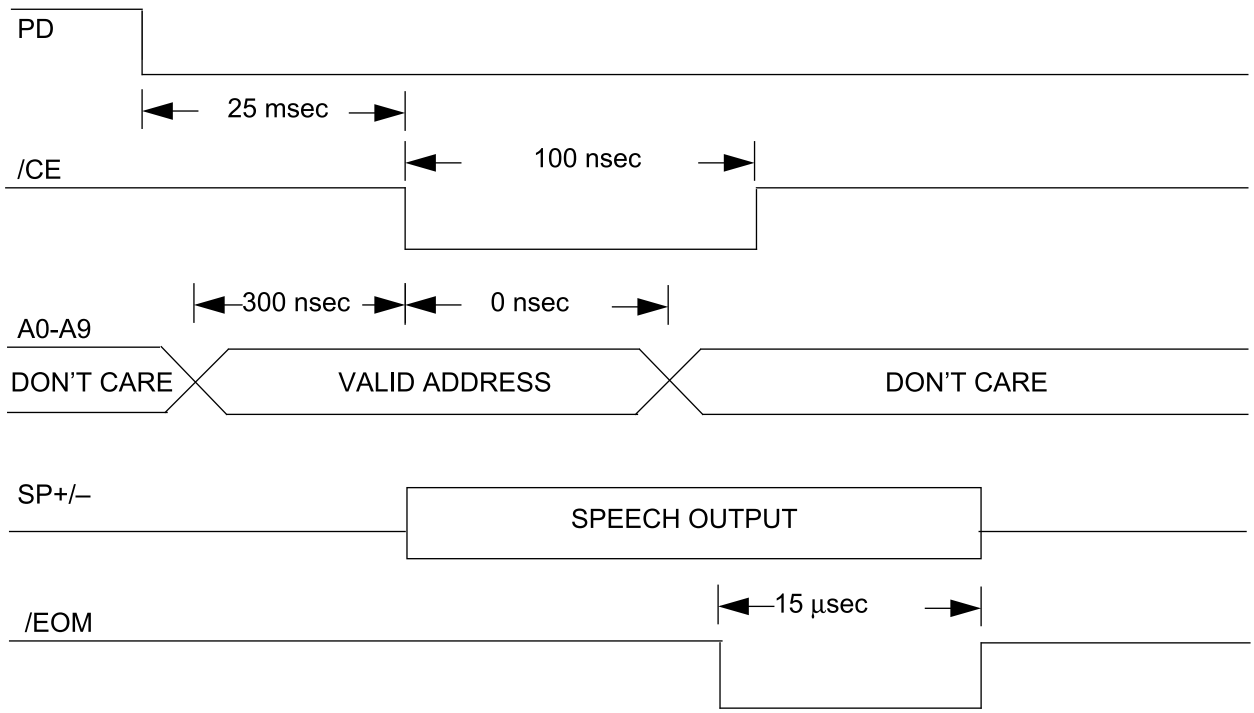


Figure 6. ISD2560 Control Signals Timing Diagram

The audible thermometer's main program flow is:

1. Initialize the MC68HC(7)05J1A's I/O ports.
2. Put the MC68HC(7)05 into low-power stop mode.
3. Wait for the user to press the pushbutton.
4. Acquire a temperature reading from the DS1820.
5. Output the reading to the ISD2560.
6. Return to stop mode and wait for the user to press the pushbutton.

Consult **Main Program Flowchart** for a detailed flowchart of the main program's operation.

After initializing the MC68HC(7)05J1A's I/O ports, the MCU is placed in stop mode. Pressing the pushbutton generates an MCU IRQ interrupt that wakes the processor out of stop mode. The processor then uses low-level driver routines to start a DS1820 temperature acquisition and conversion operation and read a 9-bit data word from the DS1820. (Consult Appendix B for a flowchart of the temperature acquisition routine.) If an error occurs during the acquisition of the word, the thermometer is placed into stop mode. Otherwise, the MC68HC(7)05J1A processes the word and determines the sequence of phrases to be output by the ISD2560. The processor then finds the address of each phrase from a series of tables. The address of each phrase is placed in the proper order in a phrase buffer. (Consult Appendix C for the flowchart of the audio processing routine.) The MCU then uses the ISD2560 low-level routines to output the sequence of phrases whose addresses are in the phrase buffer. After outputting the phrase sequence, the MCU returns to stop mode.

Summary

The ISD2560 1000 and 2500 series of voice record/playback devices permit the implementation of cost-effective, voice-based user interfaces in products based on Freescale's M68HC05 microcontrollers. The devices are designed with a microcontroller interface that easily interfaces with even the simplest member of the M68HC05 Family.

Bibliography

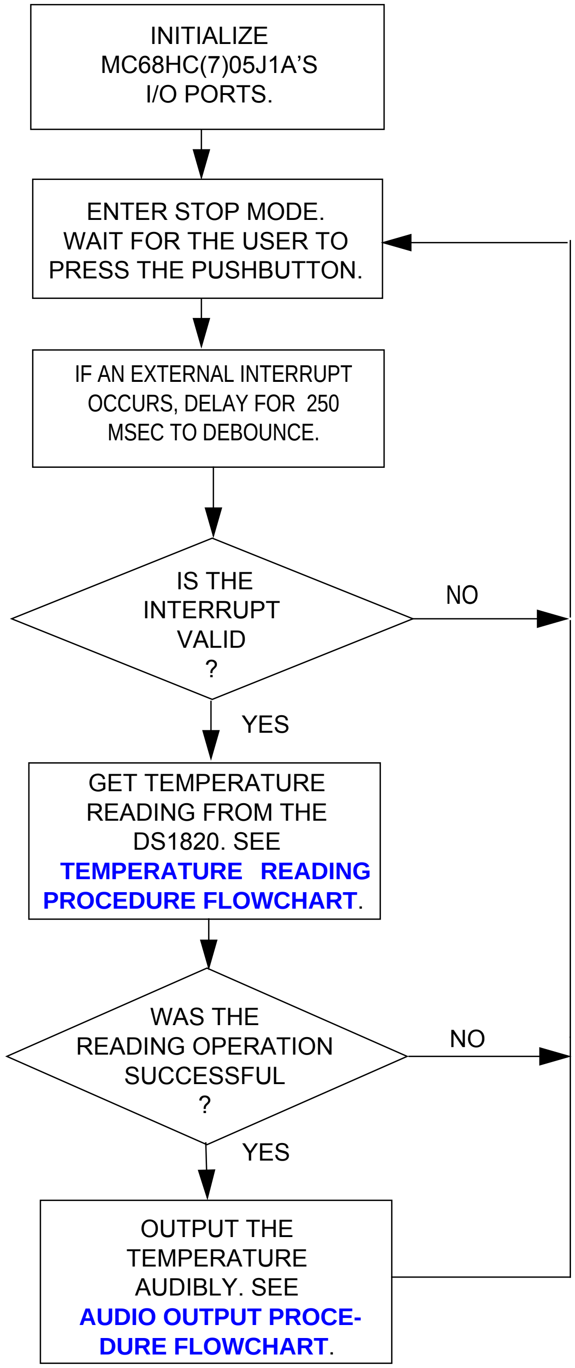
Freescale MC68HC705J1A Technical Data

ISD Information Storage Devices ISD2500 Series Preliminary Data Sheet

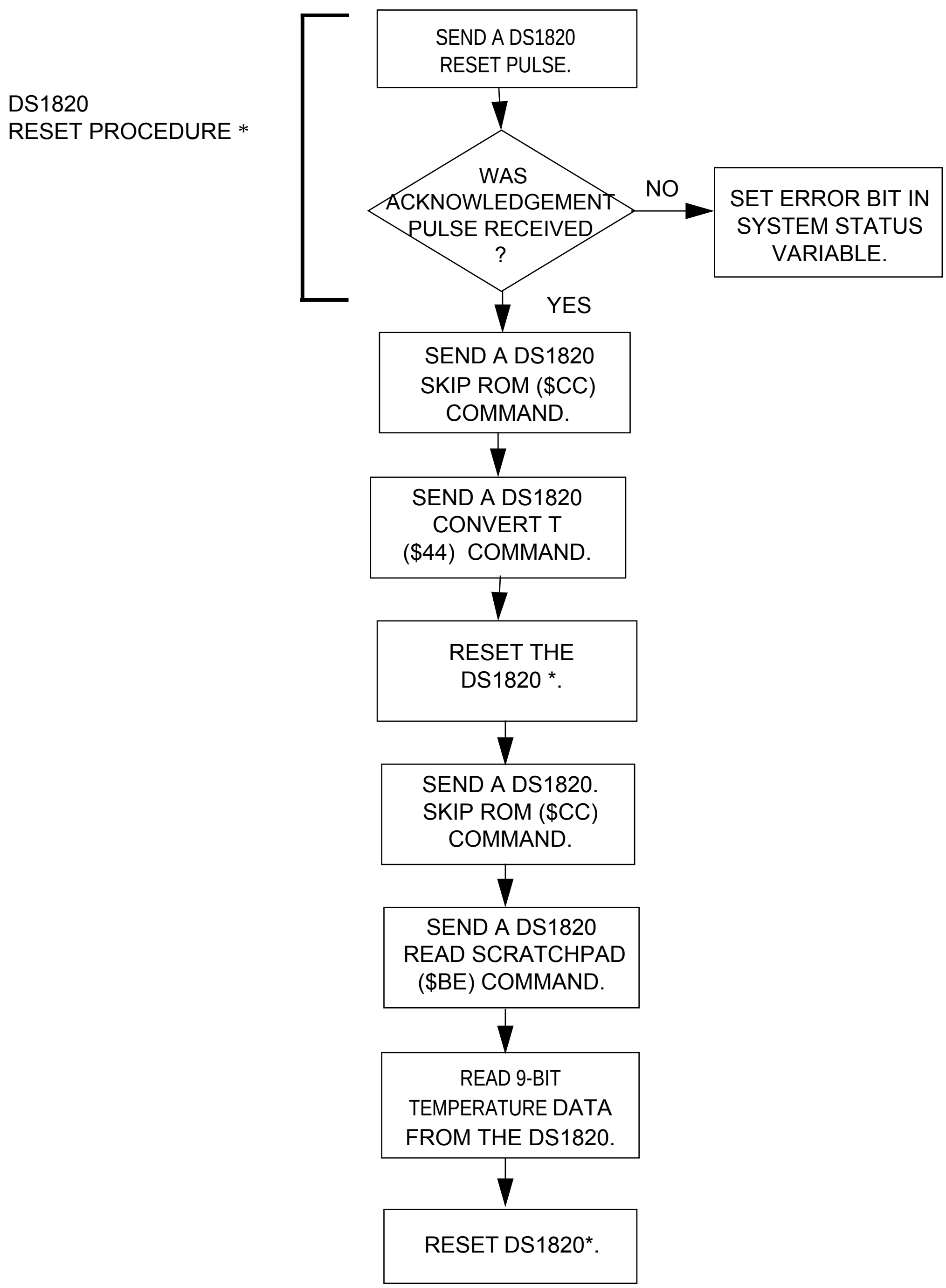
Design Manual for the ISD1000A Family

Dallas Semiconductor DS1820 One-Wire Digital Thermometer Data Sheet

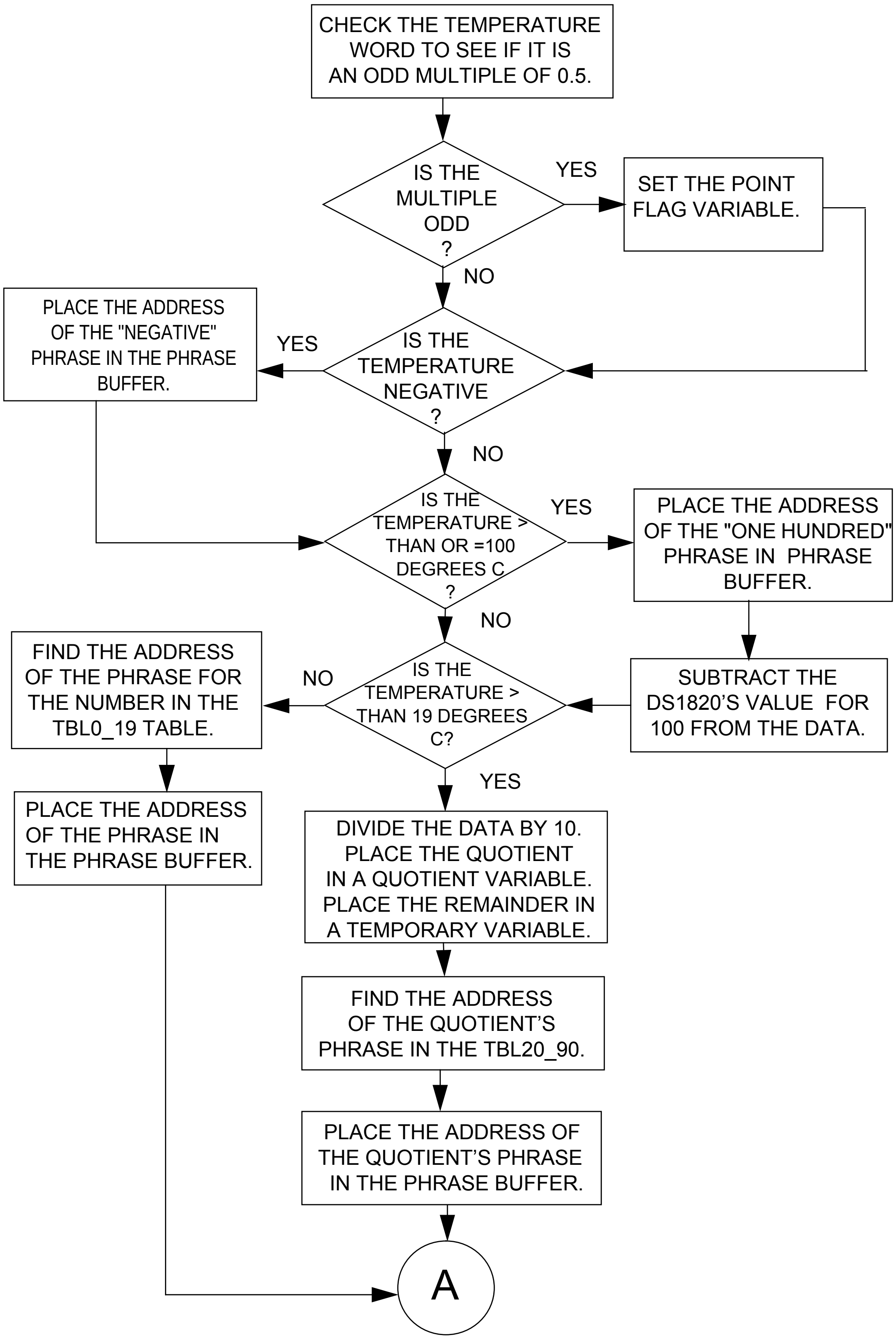
Main Program Flowchart



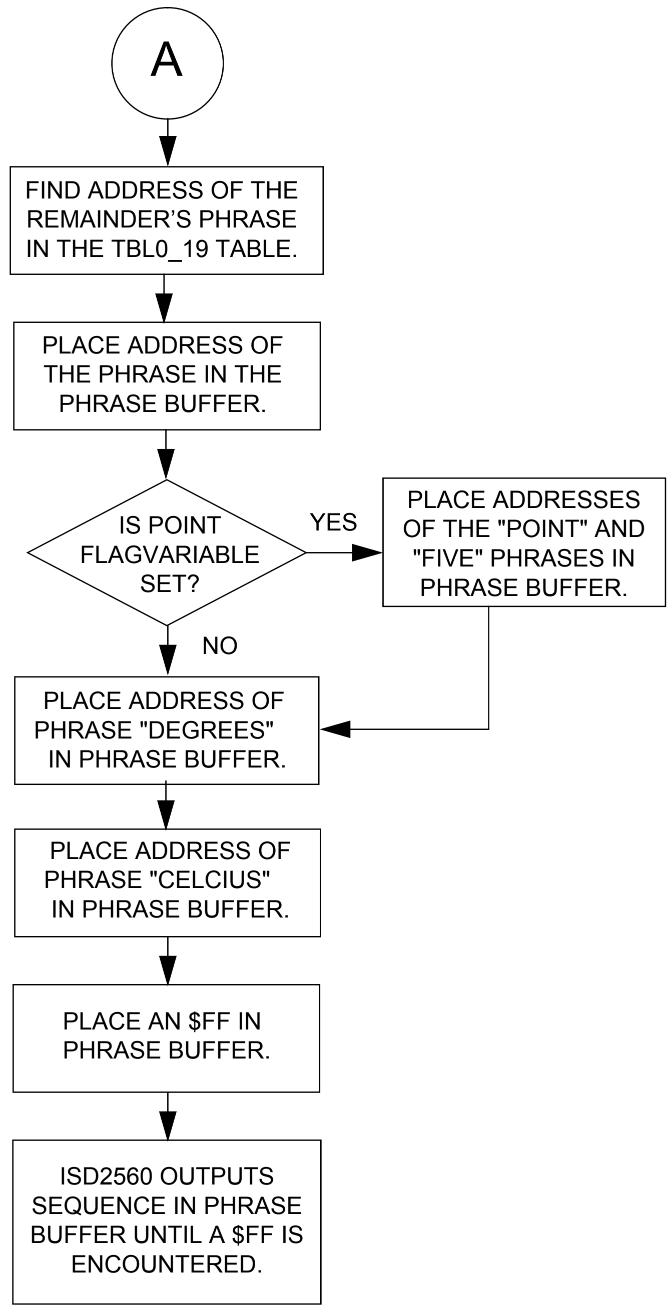
Temperature Reading Procedure Flowchart



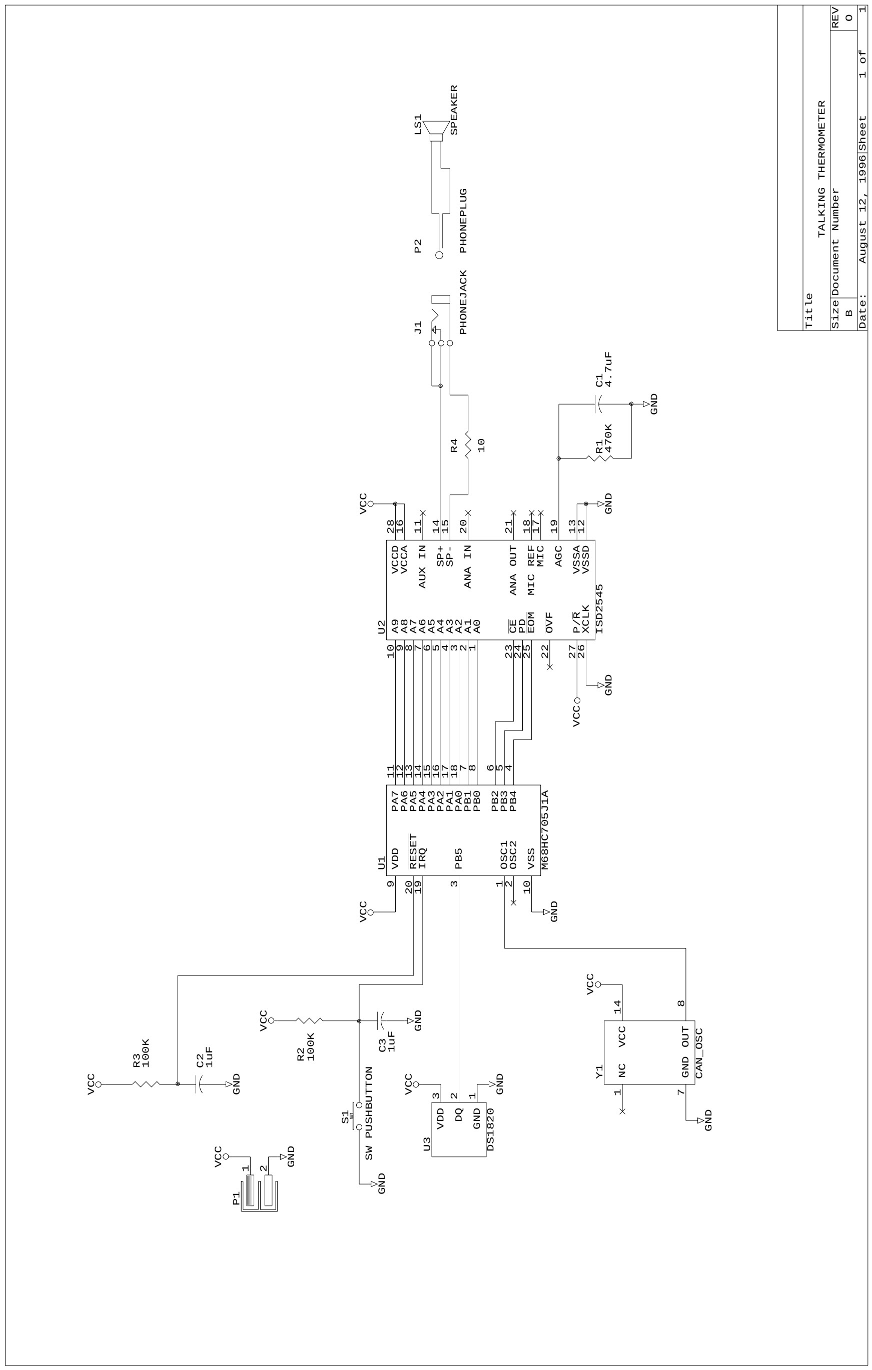
Audio Output Procedure Flowchart



Audio Output Procedure Flowchart (Continued)



Audible Thermometer Schematics



Source Code

THERMO.ASM

```
********* SYSTEM EQUATES *********

PORTA   EQU     $00     ; Port A register
PORTB   EQU     $01     ; Port B register
DDRA    EQU     $04     ; Port A Data Direction register
DDRB    EQU     $05     ; Port B Data Direction register
ERROR   EQU     0       ; Error Bit
DQ      EQU     5       ; 1820 DQ signal
DQ_CTRL EQU     5
SKIPROM EQU     $CC     ; 1820 Skip ROM command byte
CONVERT EQU     $44     ; 1820 Temperature Convert command byte
READRAM EQU     $BE     ; 1820 Read RAM command byte
CE      EQU     $02     ; ISD2560 chip enable bit
PD      EQU     $03     ; ISD2560 powerdown bit
EOM     EQU     $04     ; ISD2560 end of message bit
DDRAMSK EQU     $FF     ; Port A Data Direction register mask
DDRBMSK EQU     $2F     ; Port B Data Direction register mask
PORTAMSK EQU    $00     ; Port A mask
PORTBMSK EQU    $2C     ; Port B mask
POSITIVE_SIGN   EQU     $00    ; MSB of a positive temperature reading
NEGATIVE_SIGN   EQU     $FF    ; MSB of a neagtive temperature reading
```

```
POSITIVE_LIMIT  EQU     $FA    ; The highest LSB for a positive temperature.
NEGATIVE_LIMIT  EQU     $92    ; The lowest LSB for a negative temperature.


********* VARIABLES *********

                ORG $C0

SYS_STATUS      DS      1       ; System status variable
TEMP_HI         DS      1       ; Stores the temperature reading high byte
TEMP_LO         DS      1       ; Stores the temperature reading low byte
TEMP            DS      1       ; Temporary storage space
TEMPA           DS      1       ; Register A tempoary storage space
TEMPX           DS      1       ; Register X temporary storage space

RAW_TEMP        EQU   TEMP_HI   ; Storage space for converted reading
PHRASE_BUFFER   DS     $11      ; Stores addresses of phrases to be output
POINT_FLAG      DS      1       ; Flag indicating a .5 increment in temperature
QUOTIENT        DS      1       ; Storage space for the result of division
PHRASE_POINTER  DS      1       ; Pointer to current address in phrase buffer

                ORG $300

START:          JSR     INITIALIZE      ; Initialize J1A's I/O ports
WAIT4INT        STOP                    ;Stop
                BRA     WAIT4INT

IRQ_INT:        CLR     SYS_STATUS      ; Clear the error bit
                JSR     DEBOUNCE        ; Debounce the activation switch
                BRSET   ERROR,SYS_STATUS,IRQ_INT_EXIT ; If the error bit is
                                        ; set, the exit routine
                JSR     GET_TEMP        ; Get a temperature reading from the 1820
                BRSET   ERROR,SYS_STATUS,IRQ_INT_EXIT ; If the error bit is
                                        ; set, the exit routine
                JSR     FORM_PHRASE     ; Form table of addresses of the phrases to
                                          be output
                JSR     OUTPUT_TEMP     ; Audibly output temperature
IRQ_INT_EXIT    BCLR    ERROR,SYS_STATUS ; Clear the error bit
                RTI
```

```
***************************************************************************
*                                                                         *
*  Function Name: OUTPUT_TEMP                                             *
*  Function Inputs: None                                                  *
*  Functions Outputs: None                                                *
*                                                                         *
*  Purpose: This function outputs the contents of the                     *
*  phrase_buffer to the ISD2560 which outputs them                        *
*  audibly.                                                               *
*                                                                         *
***************************************************************************

OUTPUT_TEMP:  BCLR  PD,PORTB            ; Take the ISD2560 out of powerdown mode.
              LDX   #PHRASE_BUFFER      ; Point to the phrase buffer.
OUT_PHRASE:   LDA   PORTB
              AND   #$FC
              ORA   ,X
              STA   PORTB
              INCX
              LDA   ,X          ; Put the address of the next phrase to
              STA   PORTA       ; be output on the address bus of the ISD2560
              INCX
              BCLR  CE,PORTB    ; Pulse the ISD2560's chip enable pin to start
              BSET  CE,PORTB    ; outputting the current phrase.
EOM_H_WAIT:   BRSET EOM,PORTB,EOM_H_WAIT ; Wait for the ISD2560's End of Message
EOM_L_WAIT:   BRCLR EOM,PORTB,EOM_L_WAIT ; pulse before continuing
              LDA   ,X          ; Look for the end of the phrases to be output
              CMP   #$FF        ; if it is found exit the routine. Otherwise
              BNE   OUT_PHRASE  ; continue outputting phrases.
              BSET  PD,PORTB    ; Put the ISD2560 into powerdown mode.
              RTS
```

```
************************************************************************
*                                                                      *
*  Function Name: FORM_PHRASE                                          *
*  Function Inputs: None                                               *
*  Functions Outputs: None                                             *
*                                                                      *
*  Purpose: This function converts the temperature read               *
*  from the 1820 to the addresses of the phrases in                    *
*  the ISD2560 that match the individual digits in the                 *
*  reading. These addresses are stored in the phrase                   *
*  buffer.                                                             *
*                                                                      *
************************************************************************


FORM_PHRASE:  CLR   POINT_FLAG  ; Check to see if the temperature reading is a
                                ; a .5 increment, if it is set the POINT_FLAG.
              BRCLR 0,(RAW_TEMP+1),NOT_POINT
              INC   POINT_FLAG
NOT_POINT:    LDX   #PHRASE_BUFFER
              LDA   RAW_TEMP    ; Check to see if the temperature is negative
              BEQ   NOT_NEG      ;  if it is, place the address of the "Negative"
              LDA   NEG_ADDR     ;  phrase at the start of the phrase buffer.
Otherwise
              STA   ,X           ;  convert the temperature into its positive
                                   equivalent.
              INCX
              LDA   (NEG_ADDR+1)
              STA   ,X
              INCX
              COM   (RAW_TEMP+1)
              INC   (RAW_TEMP+1)
NOT_NEG:      LSR   (RAW_TEMP+1) ; Check for the temperature being lower than 100
degrees
              LDA   (RAW_TEMP+1) ; Celcius.
              CMP   #$64
              BLO   BELOW_100
              SUB   #$64
              STA   (RAW_TEMP+1)
              LDA    HUNDRED_ADDR ; If the temperature is greater than or equal to
                                  ; 100 degrees
              STA    ,X           ; put the address of the "One hundred" phrase in
                                    the phrase
             INCX                ; buffer and subtract the equivalent value of 100
                                 from the value.
             LDA   (HUNDRED_ADDR+1)
             STA   ,X
             INCX
             LDA   (RAW_TEMP+1)
             BEQ   POINT
```

```
BELOW_100:       LDA    (RAW_TEMP+1) ; Check to see if the remaining temperature value
                                       is less than 20
                 CMP   #$14           ; degrees. If it is, search for it in the TB0_19
                                       table.
                 BLO  BELOW_20        ; Otherwise divide the data by ten. Store the
                                       quotient in the
                 CLR  QUOTIENT        ; quotient variable and the remainder in
                                       (RAW_TEMP+1).
                 SUB    #$14
    DIV10        CMP    #$A
                 BLO    DIV_DONE
                 INC    QUOTIENT
                 SUB    #$A
                 BRA    DIV10
    DIV_DONE     STA    (RAW_TEMP+1)
                 ASL    QUOTIENT
                 STX    PHRASE_POINTER ; Find the address of the quotient's phrase in
                 LDX    QUOTIENT       ;the TBL20_90 table and store it in the phrase
    buffer.
                 LDA    TBL20_90,X
                 INCX
                 STX    TEMP
                 LDX    PHRASE_POINTER
                 STA    ,X
                 INCX
                 STX    PHRASE_POINTER
                 LDX    TEMP
                 LDA    TBL20_90,X
                 LDX    PHRASE_POINTER
                 STA    ,X
                 INCX
                 LDA    (RAW_TEMP+1)
                 BEQ    POINT
    BELOW_20     LDA    (RAW_TEMP+1) ; Find the address of the remainder's phrase in
                        the
                 ASLA                ; TBL0_19 table and store it in the phrase
                                       buffer.
                 STX    PHRASE_POINTER
                 TAX
                 LDA    TBL0_19,X
                 INCX
                 STX    TEMP
                 LDX    PHRASE_POINTER
                 STA    ,X
                 INCX
                 STX    PHRASE_POINTER
                 LDX    TEMP
                 LDA    TBL0_19,X
                 LDX    PHRASE_POINTER
                 STA    ,X
                 INCX
```

```
POINT         TST   POINT_FLAG      ; If the temperature is a .5 increment reading
              BEQ  END_RAWTEMP      ; load the phrase buffer with the addresses for
                                      the
               LDA   POINT_ADDR     ; "Point" and "Five" phrases.
               STA   ,X
               INCX
               LDA   (POINT_ADDR+1)
               STA   ,X
               INCX
               LDA   FIVE_ADDR
               STA   ,X
               INCX
               LDA   (FIVE_ADDR+1)
               STA   ,X
               INCX
END_RAWTEMP    LDA   DEGREE_ADDR     ; Load the phrase buffer with the address for
               STA   ,X              ; the "Degrees" phrase.
               INCX
               LDA   (DEGREE_ADDR+1)
               STA   ,X
               INCX
                LDA    CELCIUS_ADDR  ; Load the phrase buffer with the address for
               STA   ,X             ; the "Celcius" phrase.
               INCX
               LDA   (CELCIUS_ADDR+1)
               STA   ,X
               INCX
               CLR   ,X
               DEC   ,X
               RTS

*********************************************************************************
*                                                                               *
*  Function Name: INITIALIZE                                                    *
*  Function Inputs: None                                                        *
*  Functions Outputs: None                                                      *
*                                                                               *
*  Purpose: This function configures PORT A and PORT B                          *
*  and their data direction registers.                                          *
*                                                                               *
*********************************************************************************

INITIALIZE     LDA   #DDRAMSK
               STA   DDRA
               LDA   #PORTAMSK
               STA   PORTA
               LDA   #DDRBMSK
               STA   DDRB
               LDA   #PORTBMSK
               STA   PORTB
               RTS
```

```
*************************************************************************
*                                                                       *
*  Function Name: GET_TEMP                                              *
*  Function Inputs: None                                                *
*  Functions Outputs: None                                              *
*                                                                       *
*  Purpose: This function performs the required reads and               *
*  writes to the 1820 to perform a temperature conversion               *
*  and acquisition. The temperature read is returned in                 *
*  TEMP variable.                                                       *
*                                                                       *
*************************************************************************

GET_TEMP        JSR     RESET_1820                 ; Reset the 1820.
                BRSET   ERROR,SYS_STATUS,GET_ERROR
                LDA     #SKIPROM        ; Send the 1820's SKIP ROM command.
                STA     TEMP
                JSR     WRITE_1820
                LDA     #CONVERT        ; Send the 1820's CONVERT T command.
                STA     TEMP
                JSR     WRITE_1820
READ_LOOP       JSR     READ_1820
                LDA     TEMP
                CMP     #$FF
                BNE     READ_LOOP
                JSR     RESET_1820      ; Reset the 1820.
                BRSET     ERROR,SYS_STATUS,GET_ERROR ; If the reset fails set the
                LDA     #SKIPROM        ; error bit and exit the routine.
                STA     TEMP            ; Send the 1820's SKIP ROM command.
                JSR     WRITE_1820
               LDA       #READRAM        vccccccccc; Read the 1820's RAM to get
                                            the temperature
                STA     TEMP            ; reading.
                JSR     WRITE_1820
                JSR     READ_1820
                LDA     TEMP
                STA     TEMP_LO
                JSR     READ_1820
                LDA     TEMP
                STA     TEMP_HI
                CMP     #POSITIVE_SIGN   ; Check for an invalid positive
                BEQ     CHK_POSITIVE     ; data value.
                CMP     #NEGATIVE_SIGN   ; Check for an invalid negative
                BNE     GET_ERROR        ; data value.
                LDA     TEMP_LO
                CMP     #NEGATIVE_LIMIT
                BLO     GET_ERROR
                BRA     GET_EXIT
```

```
CHK_POSITIVE     LDA     TEMP_LO
                 CMP     #POSITIVE_LIMIT
                 BLS     GET_EXIT
GET_ERROR        BSET    ERROR,SYS_STATUS ; Set the error bit if an error
GET_EXIT         JSR     RESET_1820       ; occurs.
                 RTS

*********************************************************************************
*                                                                               *
*  Function Name: RESET_1820                                                    *
*  Function Inputs: None                                                        *
*  Functions Outputs: None                                                      *
*                                                                               *
*  Purpose: This function resets the 1820. If the 1820                          *
*  resets properly, it will return a response pulse. If                         *
*  a pulse is not received, the error bit is set in                             *
*  system status.                                                               *
*                                                                               *
*********************************************************************************

RESET_1820       STA     TEMPA           ; Save the CPU registers
                 STX     TEMPX
                 BSET    DQ,PORTB        ; Send a reset pulse to
                 BSET    DQ_CTRL,DDRB    ; the 1820
                 BCLR    DQ,PORTB
                 JSR     DELAY_500uS
                 BSET    DQ,PORTB
                 BCLR    DQ_CTRL,DDRB    ; Set the J1A to receive the
                 JSR     DELAY_100uS     ; response pulse from the 1820
                 BRSET   DQ,PORTB,RESET_ERR ; If the start of the pulse
                 JSR     DELAY_500uS     ;is not received, handle the error
                 BRSET   DQ,PORTB,RESET_EXIT
RESET_ERR        BSET    ERROR,SYS_STATUS ;Set the error bit
RESET_EXIT       BSET    DQ,PORTB        ; Set the J1A for transmission
                 BSET    DQ_CTRL,DDRB
                 LDA     TEMPA           ; Restore CPU registers
                 LDX     TEMPX
                 RTS
```

```
*****************************************************************************
*                                                                           *
*  Function Name: WRITE_1820                                                *
*  Function Inputs: None                                                    *
*  Functions Outputs: None                                                  *
*  Purpose: This function writes the data stored in the                     *
*  TEMP variable to the 1820.                                               *
*                                                                           *
*****************************************************************************

WRITE_1820       STA     TEMPA           ; Save CPU registers.
                 STX     TEMPX
                 LDX     #8              ; Load X with count.
WRITE_SHIFT      LSR     TEMP            ; Shift out the bit to be sent
                 BCS     WRITE_ONE
WRITE_ZERO       BCLR    DQ,PORTB        ; Send a zero to the 1820
                 JSR     DELAY_80uS
                 BSET    DQ,PORTB
                 BRA     DEC_WRITE
WRITE_ONE        BCLR    DQ,PORTB        ; Send a one to the 1820
                 NOP
                 NOP
                 NOP
                 BSET    DQ,PORTB
                 JSR     DELAY_80uS
DEC_WRITE        DECX
                 BNE     WRITE_SHIFT
                 LDA     TEMPA           ; Restore CPU registers
                 LDX     TEMPX
                 RTS

*****************************************************************************
*                                                                           *
*  Function Name: READ_1820                                                 *
*  Function Inputs: None                                                    *
*  Functions Outputs: None                                                  *
*                                                                           *
*  Purpose: This function reads data from the 1820 and                      *
*  returns the data in the TEMP variable.                                   *
*                                                                           *
*****************************************************************************

READ_1820        STA     TEMPA           ; Save CPU registers
                 STX     TEMPX
                 LDX     #8              ; Load X registers with count
```

```
READ_BIT         BSET    DQ,PORTB          ; Set up the DQ line for read
                 BSET    DQ_CTRL,DDRB
                 BCLR    DQ,PORTB
                 NOP
                 NOP
                 NOP
                 NOP
                 NOP
                 BCLR    DQ_CTRL,DDRB      ; Set the DQ line to receive data
                 BRSET   DQ,PORTB,READ_ONE ; Read bit
                 CLC
                 BRA     READ_SHIFT
READ_ONE         SEC
READ_SHIFT       ROR     TEMP              ; Rotate the bit in the TEMP variable
                 JSR     DELAY_80uS
                 DECX
                 BNE     READ_BIT
                 BSET    DQ,PORTB
                 BSET    DQ_CTRL,DDRB
                 LDA     TEMPA             ; Restore CPU registers
                 LDX     TEMPX
                 RTS

*********************************************************************************
*                                                                               *
*  Function Name: DEBOUNCE                                                      *
*  Function Inputs: None                                                        *
*  Functions Outputs: None                                                      *
*                                                                               *
*  Purpose: This function debounces the pushbutton switch.                      *
*                                                                               *
*********************************************************************************

DEBOUNCE         LDX     #$FF
DEBOUNCE_LOOP    JSR     DELAY_500uS
                 DECX
                 BNE     DEBOUNCE_LOOP
                 BIL     DEBOUNCE_EXIT     ; If the interrupt is valid, exit
                                           ; the routine
                 BSET    ERROR,SYS_STATUS  ; If the interrupt is invalid, set
                                           ; the error bit and exit
DEBOUNCE_EXIT    RTS
```

```
*****************************************************************************
*                                                                           *
*  Function Inputs: None                                                    *
*  Functions Outputs: None                                                  *
*                                                                           *
*  Purpose: This function provides delays.                                  *
*                                                                           *
*****************************************************************************

DELAY_80uS       LDA     #$0C
                 BRA     DELAY_LOOP
DELAY_100uS      LDA     #$0F
                 BRA     DELAY_LOOP
DELAY_500uS      LDA     #$52
                 BRA     DELAY_LOOP
DELAY_LOOP       NOP
                 NOP
                 NOP
                 DECA
                 BNE     DELAY_LOOP
                 RTS



*****************************************************************************
*                                                                           *
*                PHRASE ADDRESS TABLE                                       *
*                                                                           *
*****************************************************************************

               org $700

TBL0_19:       DW  $0000        ; Address for the phrase "Zero".
               DW  $0010        ; Address for the phrase "One".
               DW  $0020        ; Address for the phrase "Two".
               DW  $0030        ; Address for the phrase "Three".
               DW  $0040        ; Address for the phrase "Four".
FIVE_ADDR:     DW  $0050        ; Address for the phrase "Five".
               DW  $0060        ; Address for the phrase "Six".
               DW  $0070        ; Address for the phrase "Seven".
               DW  $0080        ; Address for the phrase "Eight".
               DW  $0090        ; Address for the phrase "Nine".
               DW  $00A0        ; Address for the phrase "Ten".
               DW  $00B0        ; Address for the phrase "Eleven".
               DW  $00C0        ; Address for the phrase "Twelve".
               DW  $00D0        ; Address for the phrase "Thirteen".
               DW  $00E0        ; Address for the phrase "Fourteen".
               DW  $00F0        ; Address for the phrase "Fifteen".
               DW  $0100        ; Address for the phrase "Sixteen".
               DW  $0110        ; Address for the phrase "Seventeen".
```

```
                DW   $0120          ; Address for the phrase "Eighteen".
                DW   $0130          ; Address for the phrase "Nineteen".

TBL20_90:       DW   $0140          ; Address for the phrase "Twenty".
                DW   $0150          ; Address for the phrase "Thirty".
                DW   $0160          ; Address for the phrase "Forty".
                DW   $0170          ; Address for the phrase "Fifty".
                DW   $0180          ; Address for the phrase "Sixty".
                DW   $0190          ; Address for the phrase "Seventy".
                DW   $01A0          ; Address for the phrase "Eighty".
                DW   $01B0          ; Address for the phrase "Ninety".

HUNDRED_ADDR:   DW   $01C0          ; Address for the phrase "One Hundred".

POINT_ADDR:     DW   $01D0          ; Address for the phrase "Point".

DEGREE_ADDR:    DW   $01E0          ; Address for the phrase "Degree".

NEG_ADDR:       DW   $01F0          ; Address for the phrase "Negative".

CELCIUS_ADDR:   DW   $0200          ; Address for the phrase "Celcius".


                ORG      $7FA
                DW       IRQ_INT

                ORG      $7FE
                DW       START

                END
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com