



APPLICATION NOTE 3769

## Building a 1-Wire Temperature Logger Using the MAXQ3210

*Abstract: This application note describes a temperature logging application using the MAXQ3210 microcontroller and the DS1822 1-Wire Digital Thermometer. Temperature values are logged into the MAXQ3210's internal EEPROM for later download and analysis.*

### Overview

Environmental monitoring is often performed by small, flexible microcontrollers. In applications where the power and memory capacity of a personal computer would be largely wasted, a dedicated microcontroller can be used to communicate with sensors measuring temperature, humidity, or other environmental characteristics, and then to read and store the measured values. For more flexibility, these microcontrollers can be networked together into a larger system, passing their measurements upward to more powerful systems which, in turn, analyze and record measurements from an entire facility or installation.

This application note describes an environmental monitoring application using the low-power [MAXQ3210](#) voltage-regulator microcontroller. By adding the [DS1822](#), a digital thermometer which receives both power and communications over the single-line 1-Wire® bus, we can construct a battery-powered, nonvolatile temperature logging system that requires a minimum of components.

The [demonstration code](#), which is available for download, was written in MAXQ assembly language and compiled using the standard macro preprocessor and assembler included with the MAX-IDE development environment. The code is targeted for the MAXQ3210 Evaluation Kit board, which, in turn, should be used with the following additional components (**Figure 1**).

- Temperature sensor: DS1822 Econo 1-Wire Digital Thermometer (in TO-92 package)
- RS-232 level shifter: MAX233ACWP

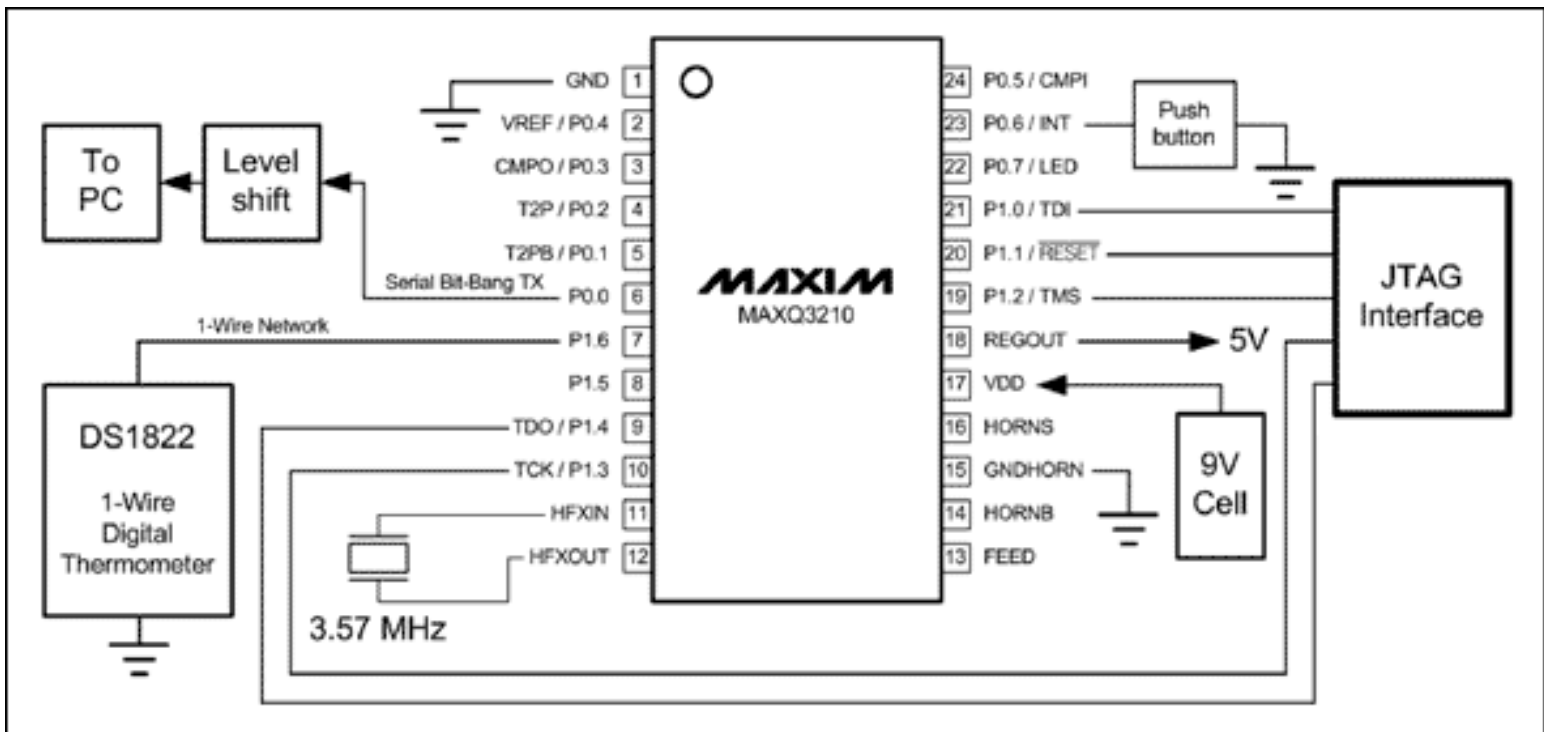


Figure 1. Components used in the MAXQ3210 1-Wire temperature logger demonstration circuit.

## Design Goals

The demonstration code performs the following tasks (**Figure 2**):

- Communicates over the 1-Wire network (bit-banged) with the DS1822 temperature sensor
- Wakes up once a minute to take a temperature measurement
- Records temperature measurements in a nonvolatile log in the MAXQ3210's internal EEPROM data memory
- Transmits the contents of the temperature log over a bit-banged serial port at 9600 baud upon power-up
- Converts temperatures to human-readable ASCII format (decimal degrees Fahrenheit) before transmitting
- Performs a master clear (erases the temperature log stored in the data EEPROM) upon request

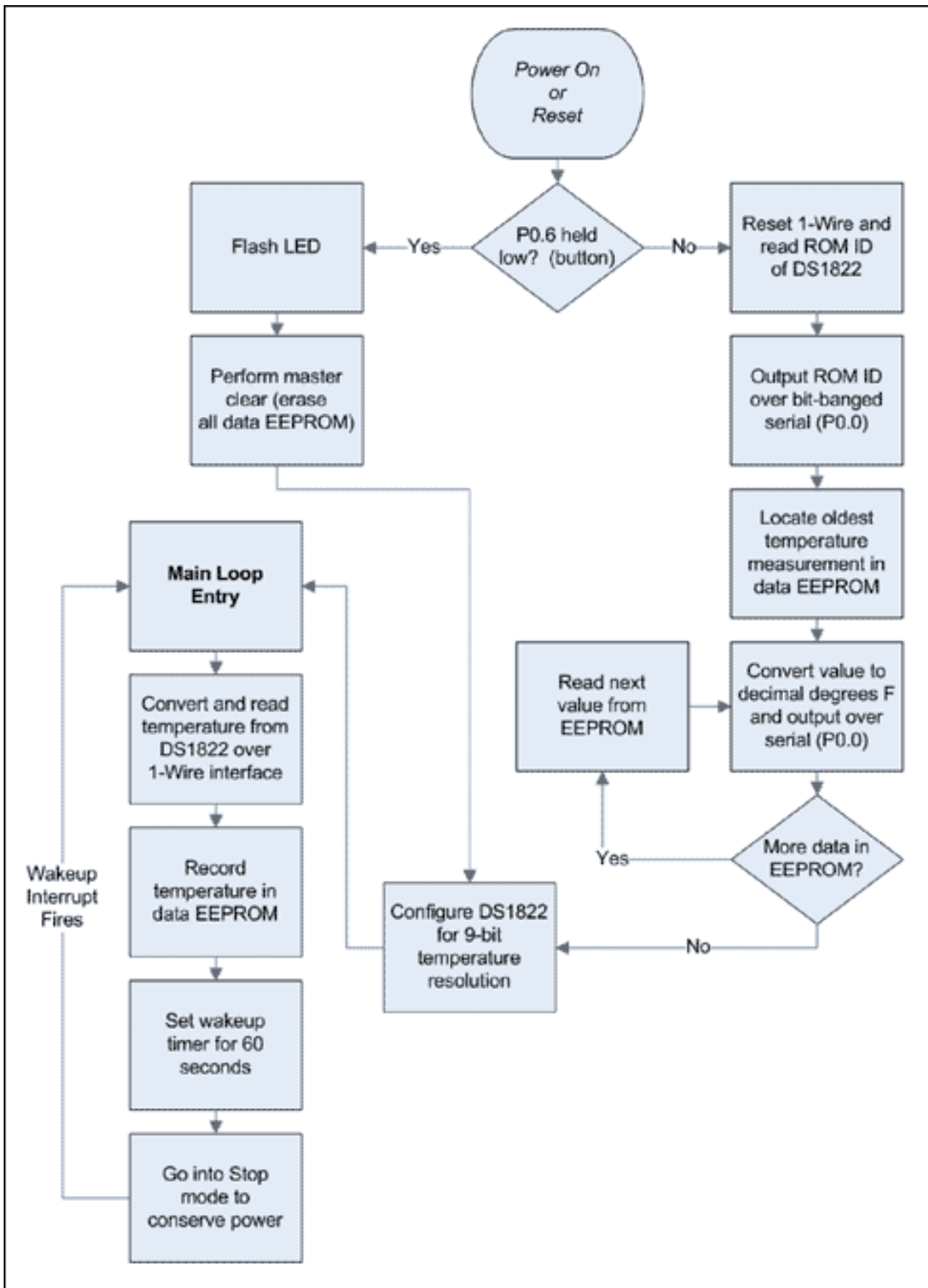


Figure 2. Execution flow of the temperature logging application.

## Why Use the MAXQ3210?

Although almost any low-power MAXQ microcontroller could be used for the heart of this demonstration, the MAXQ3210 is ideal for the temperature logging application.

- **Integrated Voltage Regulator.** The MAXQ3210 has an internal 5V regulator which allows it to run directly from a standard 9V battery. The MAXQ3210 also provides the 5V regulated output for use by other devices (up to 50mA). This feature is important because it means that, if all other devices in the design can run from 5V, no separate power-supply components are needed.
- **Low Power.** The MAXQ3210 draws minimal current, typically around 6mA, even when operating at its full 3.58MHz speed. Battery current can be reduced during periods of inactivity by shifting to a lower clock rate or even by halting the processor completely in Stop mode. The MAXQ3210's integrated 8kHz internal ring oscillator drives a long-period wakeup timer that can wake the processor from Stop mode after a preprogrammed interval up to two minutes long.
- **Internal Data EEPROM.** In case of power loss, the temperature logging application will retain its recorded data, which may have been collected over hours, days, or even weeks. The MAXQ3210 makes this simple by providing a separate 64-word EEPROM memory in data space. The 16-bit words in this memory can be written individually using a simple call to a routine in the Utility ROM; the EEPROM technology means that an erase operation is never required before writing data. If more space is required, any unused program EEPROM space can be written word-by-word in a similar manner using another Utility ROM routine. You do not need to reload the entire application to do this procedure.
- **5V Port Pins.** Like all MAXQ microcontrollers, the MAXQ3210 has flexible port pins which can operate in input, output, weak pullup, and tristate modes. The MAXQ3210 also provides a wide range of interfacing options. Because the microcontroller's pins run on a 5V rail, they can connect to 5V-powered devices directly and to lower powered devices by means of pullup resistors (operating in open-drain/tristate mode). Because only a few port pins are required for this application, using a larger microcontroller would leave most of its functionality unused.
- **Piezoelectric Horn Driver.** The piezoelectric horn function is not used in this application. Nonetheless, the ability to drive a loud, audible alarm is a requirement for many types of environmental monitoring applications. Smoke detectors and carbon monoxide sensors are typical examples. The MAXQ3210 provides a direct interface connection to a piezoelectric horn, which is extremely simple to operate from software. A single bit turns the horn on or off as needed. Depending on the horn used, the MAXQ3210 can output volume levels reaching up to 100dB.
- **Small Package.** The MAXQ3210 comes in a compact 24-pin TSSOP package.

## Driving the 1-Wire Network

Dallas Semiconductor/Maxim developed a wide range of sensors and other components that operate over the 1-Wire network interface. This interface provides both power and communications over a single wire plus ground, which means that a microcontroller can communicate with a 1-Wire sensor over a single port pin.

The 1-Wire network operates with a single master and multiple slaves (multidrop). Timing requirements are flexible, allowing all slaves to synchronize with the master at communications speeds up to 16kbps. Each 1-Wire sensor has a globally unique 64-bit ROM ID, allowing the 1-Wire master to select slaves individually and precisely regardless of their physical position on the network.

The 1-Wire line operates in an open-drain mode, where the master (and the slaves, when their output is requested) pulls the line down to ground to indicate a zero and lets it float high to indicate a one. Normally, this process is implemented by having a discrete pullup resistor attached between the line and  $V_{CC}$ . The MAXQ3210, however, has a weak pullup mode on the port pins and can simply switch the port pin back to pullup mode and let the line float high. Consequently, the MAXQ3210 requires no external resistor. Because the master and slaves only pull the line low and never pull it actively high, the 1-Wire network operates in a wired-OR configuration. This prevents line conflict in situations where multiple slaves try to transmit on the 1-Wire bus simultaneously.

To drive the 1-Wire network, the MAXQ3210 uses software to generate the following types of time slots on a single pin. Since the 1-Wire master initiates all time slots, the MAXQ3210 does not need to monitor the 1-Wire line when it is not communicating with a slave device. Refer to the DS1822 data sheet for more details on timing requirements for 1-Wire communications.

- **Reset** time slots are approximately 1ms in width. For the first half of the timeslot, the master (MAXQ3210) holds the 1-Wire line low. Halfway through the timeslot, the master releases the 1-Wire line and lets it float high. Any 1-Wire slaves present on the line respond by resetting themselves and pulling the line down during the second half of the time slot. This step generates a **presence pulse** which indicates to the master that one or more 1-Wire slaves are present on the line and ready to communicate.
- **Write** time slots are approximately 120µs in length and used by the master to transmit either 0 or 1 bits to one or more 1-Wire slaves. Both types of write time slots start with the master pulling the line low for at least a microsecond. To transmit a one, the master then releases the 1-Wire line (letting it float high) for the rest of the time slot. To transmit a zero, the master continues to hold the line low until the time slot ends.
- **Read** time slots are approximately 60µs in length and used by the master to read either 0 or 1 bits from a slave device. The time slot starts with the master pulling the line low for at least a microsecond. The master then releases the line, allowing the slave to either hold the line low (to indicate a zero) or let the line float high (to indicate a one). Midway through the time slot, the master samples the line to read the bit value from the slave.

As the MAXQ3210 runs at about three and one-half instruction cycles per microsecond (at 3.58MHz), its software can use a port pin (P1.6) and easily execute the standard 1-Wire protocol.

```

#define OWIN      M0[09h].6    ; PI1.6
#define OWOUT     M0[01h].6    ; PO1.6
#define OWDIR     M0[11h].6    ; PD1.6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Function      : Reset1Wire
;; Description   : Sends a standard speed 1-Wire reset pulse on P1.6
;;               : and checks for a presence pulse reply.
;; Inputs       : None
;; Outputs      : C - Cleared on success; set on error (no presence
;;               : pulse detected)
;; Destroys     : PSF, LC[0]
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Reset1Wire:
    move  OWDIR, #1            ; Output mode
    move  OWOUT, #0           ; Drive low
    move  LC[0], #RESET_LOW
    djnz  LC[0], $

    move  OWOUT, #1          ; Snap high
    move  LC[0], #SNAP
    djnz  LC[0], $

    move  OWDIR, #0          ; Change to weak pullup input
    move  LC[0], #RESET_PRESAMPLE
    djnz  LC[0], $

    move  C, OWIN            ; Check for presence detect

    move  LC[0], #RESET_POSTSAMPLE
    djnz  LC[0], $

    ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Function      : Write1Wire

```

```

;; Description : Writes a standard speed 1-Wire output byte on P1.6.
;; Inputs      : GRL - Byte to write to 1-Wire.
;; Outputs     : None.
;; Destroys    : PSF, AP, APC, A[0], LC[0], LC[1]
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

WritelWire:
    move    APC, #080h          ; Standard mode, select A[0] as Acc
    move    Acc, GRL
    move    OWDIR, #1          ; Output drive mode
    move    LC[1], #8          ; 8 bits to write

WritelWire_slot:
    move    OWOUT, #0          ; Drive low for start of write slot

    move    LC[0], #WRITE_PREBIT
    djnz   LC[0], $

    rrc
    jump   C, WritelWire_one

WritelWire_zero:
    move    OWOUT, #0          ; Keep the line low (zero bit)
    jump   WritelWire_next

WritelWire_one:
    move    OWOUT, #1

WritelWire_next:
    move    LC[0], #WRITE_POSTBIT
    djnz   LC[0], $          ; Finish the time slot

    move    OWOUT, #1          ; Drive back high (end of slot)
    move    LC[0], #WRITE_RECOVERY
    djnz   LC[0], $          ; Recovery time period

    djnz   LC[1], WritelWire_slot

    ret

```

The function for sending a read time slot is implemented in a similar manner. Note that all data bytes on the 1-Wire bus are transmitted least significant bit (LSB) first.

There is one additional point about implementing the 1-Wire bus on the MAXQ3210 worth mentioning. Although the pullup resistor on the 1-Wire bus can vary depending on the number of devices on the 1-Wire network, it is typically specified around 4kΩ to 5kΩ. The weak pullup resistor on the MAXQ3210's port pin, however, is closer to 50kΩ to 100kΩ. To avoid the 1-Wire bus taking an excessive amount of time to float high from a low state, the demonstration code drives a normal high state on P1.6 for a brief interval to 'snap' the bus to the high state before setting the port pin to the normal weak pullup mode. As long as this process is not done while the slave attempts to pull the bus low, it should not cause any problems on the 1-Wire bus. An alternative implementation would put a physical external pullup resistor on the 1-Wire bus and drive the port pin in standard low mode for a zero state, and tristate mode for a high state.

Note: When implementing 1-Wire networks that run for long distances or connect a large number of devices, additional considerations may apply. Refer to the following application notes for more information.

- Application note 148, "[Guidelines for Reliable 1-Wire Networks.](#)"
- Application note 937, "[Book of iButton Standards.](#)"

## Measuring Temperature with the DS1822

Although implementing the code above allows the MAXQ3210 to communicate with most 1-Wire slave devices, we will focus on the DS1822 in this application. The DS1822 is a 1-Wire slave device, providing 9-bit to 12-bit centigrade temperature measurements that can be read by the 1-Wire master. Like most 1-Wire devices, the DS1822 can be powered completely from the 1-Wire bus, a feature known as parasite power.

The DS1822 measures from  $-55^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$ , which is more than adequate for any indoor/outdoor temperature-measurement application. The resolution for the temperature reading ranges from  $0.5^{\circ}\text{C}$  for 9-bit to  $0.0625^{\circ}\text{C}$  for 12-bit operation. The maximum time required for the DS1822 to perform a temperature measurement runs from approximately 94ms for the coarsest resolution to 750ms for the finest resolution. As this application is a simple example, we select the 9-bit resolution and ignore the smallest bit ( $0.5^{\circ}\text{C}$ ). These parameters allow the entire signed 8-bit temperature value to fit in the MAXQ3210's 8-bit accumulator registers.

All 1-Wire slave devices implement a common command set which allows the 1-Wire master to determine how many slaves are present on the 1-Wire bus, read their ROM ID values, and activate them individually or as a group. Once a 1-Wire slave is activated, the master can send it additional commands specific to that type of 1-Wire device. All other slaves which have not been activated will wait until the next reset pulse occurs before beginning to monitor the 1-Wire bus again.

Because our application only involves a single 1-Wire device on the bus, we can use the simplest set of commands to access this device and free the application from tracking the device's ROM ID. When more than one device is on the bus, this ROM ID is used to differentiate between slaves. Although the DS1822's ROM ID is read by our application at one point, this is done merely for demonstration purposes.

We will implement the following 1-Wire commands. Refer to the DS1822 data sheet for details about additional commands which are supported.

- **Read ROM [33h]**. This command assumes that only a single 1-Wire slave is present on the bus. When a 1-Wire slave receives this command, it transmits its 8-byte ROM ID back to the 1-Wire master. This ID value consists of a 48-bit serial number, an 8-bit CRC, and an 8-bit family code. The family code indicates the device type. This value is 22h for the DS1822. After receiving this Read ROM command, the 1-Wire slave activates it and responds to a subsequent device-specific command.
- **Skip ROM [CCh]**. This command, which can be used with one or more 1-Wire slaves on the bus, activates all slaves regardless of their ROM ID values. When a single slave is present on the bus, this command is a simple way to activate the slave for any device-specific command without having to read that slave's ID value. When this command is used with multiple slaves, the following device-specific command must not cause the slaves to transmit data back to the master, because multiple slaves can transmit different values and cause a data collision.
- **Write Scratchpad [4Eh]**. This command is specific to the DS1822. Consequently, a Read ROM or Skip ROM command must first be used to activate the device. Following this command, the 1-Wire master transmits three additional bytes which are used to configure the operation of the DS1822, including the bit resolution for temperature conversions. Refer to the DS1822 data sheet for more details.
- **Read Scratchpad [BEh]**. This command is also specific to the DS1822. It allows the 1-Wire master to read back up to nine bytes of data from the DS1822. These bytes include the configuration registers set by the Write Scratchpad command, as well as the most recent temperature conversion value. Refer to the DS1822 data sheet for more details. For our application only the first two bytes, which contain the last temperature value converted, are of interest.
- **Convert Temperature [44h]**. This command is specific to the DS1822. When the DS1822 receives this command, it measures the temperature and converts it to a value of the selected bit resolution. This value is stored in two internal registers which may be read back by the 1-Wire master using the Read Scratchpad command.

When the Convert Temperature command is executed, the DS1822 requires more power (up to 1.5mA) than can be provided by the weak pullup on the 1-Wire line. Consequently, once this command is issued, the master must provide a **strong pullup** on the 1-Wire line until the temperature conversion is complete. During this period, no 1-Wire communication can take place on the line. The MAXQ3210 implements this simply by switching the P1.6 port pin from weak pullup mode to normal high-output mode. The MAXQ3210's port pin drivers provide more than sufficient high-current drive to meet the DS1822's operating needs.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Function      : ConvertAndReadTemp
;; Description   : Sends commands to measure temperature and read
;;                scratchpad from the DS1822.
;; Inputs       : None.
;; Outputs      : GRL - 8-bit signed temperature value, in degrees C.
;; Destroys     : PSF, AP, APC, A[0], A[1], A[2], LC[0], LC[1]
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

ConvertAndReadTemp:
    call  Reset1Wire          ; Reset the DS1822

    move  GRL, #OW_SKIP_ROM   ; Select the DS1822
    call  Write1Wire
    move  GRL, #OW_CONVERT    ; Send temp convert command
    call  Write1Wire

    move  OWDIR, #1           ; Turn on strong pullup for draw current
    move  OWOUT, #1

    move  LC[0], #55          ; About a second
delay:
    move  LC[1], #0
    djnz LC[1], $
    djnz LC[0], delay

    call  Reset1Wire          ; Conversion completed; reset again

    move  GRL, #OW_SKIP_ROM   ; Select again
    call  Write1Wire
    move  GRL, #OW_RD_SCRATCH ; Read the scratchpad values
    call  Write1Wire

    call  Read1Wire
    move  A[1], GRL           ; Temp LSB   3210xxxx
    call  Read1Wire
    move  A[2], GRL           ; Temp MSB   sssss654

    move  Acc, A[1]           ; 3210xxxx
    and   #0F0h               ; 3210----
    xchn                ; ----3210
    move  A[1], Acc
    move  Acc, A[2]           ; sssss654
    and   #00Fh               ; ----s654
    xchn                ; s654----
    or   A[1]                 ; s6543210

    move  GRL, Acc
    ret

```

## Recording Measurements in the Data EEPROM

To guard against momentary disruptions on the 1-Wire line, the demonstration code performs three temperature conversions (A, B, and C) on the DS1822 for each measurement interval. It then selects a measurement value to store based on the following:

- If all three values are the same, store that value.
- If two out of three values agree ( $A = B$ ,  $B = C$ , or  $A = C$ ), store the value of the two matching samples.
- If none of the three values agree, store the value of the sample in the middle. For example, if  $(A > B > C)$ , store value B.

The selected value is written to one word in the data EEPROM. Since the sample is only one byte long, the high byte of each word is used to indicate whether or not that entry (i.e., word) in the log is empty. This means that if the high byte is zero, the entry/word is empty. If, however, the high byte is nonzero, the low byte contains a valid temperature value. This allows the application to differentiate between empty entries and those containing 0°C temperature readings.

```
;; Two out of three majority vote, or failing that, the measurement
;; in the middle of the three.
```

```
move Acc, A[4]
cmp A[5]
jump E, recordTempA ; If (A==B), use that value
cmp A[6]
jump E, recordTempA ; If (A==C), use that value
```

```
move Acc, A[5]
cmp A[6]
jump E, recordTempB ; If (B==C), use that value
```

```
move Acc, A[4]
sub A[5]
jump S, B_greaterThan_A ; Sign is set if (A-B) is negative
```

```
;; If (A > B) {
;;   If (C > A) record A      (C > A > B)
;;   If (B > C) record B,    (A > B > C)
;;   else record C          (A > C > B)
```

A\_greaterThan\_B:

```
move Acc, A[4]
sub A[6] ; A-C
jump S, recordTempA ; Sign is set if (A-C) is negative
move Acc, A[5]
sub A[6] ; B-C
jump S, recordTempC ; Sign is set if (B-C) is negative
jump recordTempB
```

```
;; If (B > A) {
;;   If (C > B) record B      (C > B > A)
;;   If (A > C) record B,    (A > B > C)
;;   else record C          (B > C > A)
```

B\_greaterThan\_A:

```
move Acc, A[5]
sub A[6] ; B-C
jump S, recordTempB ; Sign is set if (B-C) is negative
move Acc, A[4]
sub A[6] ; A-C
jump S, recordTempC ; Sign is set if (A-C) is negative
jump recordTempB
```

recordTempA:

```
move GRL, A[4]
jump recordTemp
```

```

recordTempB:
    move    GRL, A[5]
    jump   recordTemp

recordTempC:
    move    GRL, A[6]
    jump   recordTemp

recordTemp:
    move    A[15], GRL

    move    GRL, #'@'
    call   TxCharBB
    move    GR, DP[0]
    move    GRL, GRH
    call   TxHexByteBB
    move    GRL, DP[0]
    call   TxHexByteBB

    move    GRL, #' '
    call   TxCharBB
    move    GRL, #'W'
    call   TxCharBB

    move    GRL, A[15]
    call   TxHexByteBB

    move    GRL, A[15]           ; Low byte contains temp data
    move    GRH, #055h         ; High byte marks nonzero entry
    lcall  UROM_loadData       ; Write entry to data EEPROM

    call   IncDP0_EE           ; Move to the next entry position
    move    GR, #0000h         ; Erase any data that exists
    lcall  UROM_loadData       ; Erase the oldest entry

```

The log is written in a rolling manner, running from address 020h in the data EEPROM to 05Fh and then wrapping around. After each new entry is written, the oldest log entry is erased. When transmitting the temperature log over the serial interface, the application locates the oldest entry by searching for a temperature entry with a blank entry immediately preceding it.

## Conserving Power

Because the application only records one temperature entry per minute, and because reading the DS1822's temperature and writing it to the EEPROM takes only a few seconds, the application spends most of its time inactive, waiting for that minute-long delay to elapse. Depending on the requirements of the application, this temperature-recording interval could increase much longer, to as much as five, ten, or thirty minutes between temperature readings without much additional code. To avoid unnecessarily draining battery power during these inactive periods, we need to conserve power as much as possible.

The lowest power state available to the MAXQ3210 is Stop mode. In this mode, power consumption drops into the microamp range as program execution ceases and the high-frequency crystal oscillator shuts down. Because no other components in the application circuit are active, however, we need a way for the MAXQ3210 to bring itself back out of Stop mode on a periodic basis to take temperature measurements.

This latter requirement is accomplished by the MAXQ3210's wakeup timer. This timer, which can run off a low-current 8kHz internal ring oscillator in Stop mode, wakes the microcontroller after a preprogrammed duration which can be up to two minutes long. This timing operation is ideal for our needs, as the application can set its 'alarm

clock' for a minute, go into Stop mode to conserve power, and wait for the wakeup timer to return it to active mode.

```
;; Start the wakeup timer for 60 seconds.

    move    CKCN.6, #1          ; Select ring oscillator mode
waitRing:
    move    C, CKCN.5
    jump    NC, waitRing       ; Wait for RGMD=1 (running from ring)

    move    WUT, #30000        ; 1/8kHz * 30000 * 16 = 60 seconds
    move    WUTC, #0101b       ; Start the wakeup timer (running from ring)

    move    IV, #wakeUpInt     ; Set interrupt handler for wakeup interrupt
    move    IMR.0, #1          ; Enable interrupts from module 0
    move    IC.0, #1           ; Globally enable interrupts

    move    PD0.7, #0          ; Turn off output mode for LED pin
    move    PO0.7, #1          ; Return to default state (weak pullup)

    move    CKCN.4, #1         ; Go into Stop mode, wait for wakeup int
    nop

    jump    mainLoop           ; Back for another round

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

wakeUpInt:
    move    PD0.7, #1          ; Turn on output mode for LED port pin
    move    PO0.7, #0          ; Light the LED

    move    CKCN.6, #1          ; Select ring oscillator mode
wakeUp_ring:
    move    C, CKCN.5
    jump    NC, wakeUp_ring    ; Wait for RGMD=1 (running from ring)

    move    LC[0], #4000
    djnz   LC[0], $
    move    PO0.7, #1          ; LED off
    move    LC[0], #4000
    djnz   LC[0], $
    move    WUTC, #0           ; Clear wakeup timer flag

    move    CKCN.6, #0          ; Select crystal mode
wakeUp_xtal:
    move    C, CKCN.5
    jump    C, wakeUp_xtal     ; Wait for RGMD=0 (running from crystal)

    move    GRL, #'W'
    call   TxCharBB
    move    GRL, #'U'
    call   TxCharBB
    move    GRL, #0Dh
    call   TxCharBB
    move    GRL, #0Ah
    call   TxCharBB

    reti
```

## Uploading the Temperature Log

Following each power-up or reset cycle, the application outputs its collected temperature log data back to a host system. This output is sent in a 10-bit asynchronous serial data format (1 start bit, 8 data bits, 1 stop bit) at 9600 baud. The MAXQ3210 does not have a hardware UART serial port, so the software must upload the temperature log by using a port pin. As the application only needs to transmit, not receive, this upload is simple enough to implement.

```
;;;;;;;;;;;;;
;;
;; Function      : TxCharBB
;; Description   : Transmits a 10-bit serial character (bit-banged)
;;               : over P0.0.
;; Inputs       : GRL   - Character to send
;; Outputs      : None
;; Destroys     : PSF, AP, APC, A[0], LC[0], LC[1]
;;
;;;;;;;;;;;;;
```

```
TxCharBB:
    move  APC, #080h           ; Standard mode, select A[0] as Acc
    move  Acc, GRL

    move  P00.0, #0           ; START bit low
    move  LC[0], #BITLOOP
    djnz  LC[0], $

    move  LC[1], #8           ; 8 bits
TxCharBB_bitLoop:
    rrc   C, TxCharBB_one    ; Get the next bit
    jump  C, TxCharBB_one
TxCharBB_zero:
    move  P00.0, #0
    sjump TxCharBB_next
TxCharBB_one:
    move  P00.0, #1
TxCharBB_next:
    move  LC[0], #BITLOOP
    djnz  LC[0], $
    djnz  LC[1], TxCharBB_bitLoop

    move  P00.0, #1           ; STOP bit high
    move  LC[0], #BITLOOP
    djnz  LC[0], $
    move  LC[0], #BITLOOP
    djnz  LC[0], $

    ret
```

Converting the temperature values from signed binary, 8-bit Celsius values into human readable ASCII format in Fahrenheit degrees involves a bit more code, but is straightforward enough. The conversion from binary into decimal is performed using BCD (Binary Coded Decimal) arithmetic at the same time as the conversion from Celsius to Fahrenheit.

```
move  GR, @DP[0]           ; Get the current entry
move  Acc, GRH              ; Check the high byte
jump  Z, endOutput         ; If it's zero we're done
move  A[15], GRL           ; Save the low byte (temp value)
```

```

move  A[7], #0           ; Hundreds = 0
move  A[6], #0           ; Tens     = 0
move  A[5], #0           ; Ones     = 0
move  A[4], #0           ; Tenths   = 0

move  A[3], #0           ; Add 01.8 per degree C
move  A[2], #1
move  A[1], #8

move  Acc, A[15]         ; s6543210
jump  S, tempNegC

tempPosC:
move  GRL, #'+'
jump  Z, tempPrint

move  LC[0], Acc
tempPosC_loop:
call  AddBCD
djnz  LC[0], tempPosC_loop

move  A[3], #3
move  A[2], #2
move  A[1], #0           ; Add 32.0
call  AddBCD

jump  tempPrint

tempNegC:
move  GRL, #'-'
neg
jump  Z, tempPrint       ; Negative zero
jump  S, tempPrint       ; -128 is outside the sensor range anyhow

move  LC[0], Acc
tempNegC_loop:
call  AddBCD
djnz  LC[0], tempNegC_loop

move  A[3], #3
move  A[2], #2
move  A[1], #0           ; Subtract 32.0
call  SubBCD

jump  NC, tempPrint
move  GRL, #'+'         ; Back to positive again
jump  tempPrint

tempPrint:
call  TxCharBB          ; Print plus/minus sign
call  TxTempBB          ; Print temperature value + newline

call  IncDP0_EE         ; Move to the next entry

```

Because the port-pin output from the MAXQ3210 is simply at a 5V level, it must be level-translated by an external component (such as the MAX233ACWP) before being connected to a PC COM serial port. Once this has been done, the output from the application can be received using any standard terminal emulator program.

RST  
DS1822 Detected : 22A9CC15000000E5

+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 59.0  
+ 62.6  
+ 69.8  
+ 59.0  
+ 55.4  
+ 55.4  
+ 55.4  
+ 55.4  
+ 55.4  
+ 55.4  
+ 55.4  
+ 57.2  
+ 55.4  
+ 55.4  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2  
+ 57.2

## Extending the Application

The demonstration application as written only consumes about 60% to 70% of the MAXQ3210's 1k x 16 (1024 word) EEPROM program space. The application could easily be optimized to take as little as 50% of the program memory. With the application's core functionality developed, numerous additional features could be added to expand it into a full-fledged environmental monitoring system.

- **Multiple Sensors.** The 1-Wire routines could easily be extended to cover multiple DS1822 temperature sensors, either on individual port pins (1 device per port pin) or grouped together on a single line (multidrop). The multidrop configuration is more complex, but would allow a much larger number of devices to be linked to the MAXQ3210.
- **Additional Sensor Types.** The application could interface to several different 1-Wire sensors to measure characteristics such as humidity (the DS1923 Temperature/Humidity Logger), physical switches (the DS2401 Silicon Serial Number), or generic sensors using analog-to-digital conversion (the DS2450 1-Wire Quad A/D Converter). Please see the 1-Wire/iButton® product page on the [Maxim IC web site](#) for more details.
- **Audible Alert.** Because the MAXQ3210 has a built-in piezoelectric horn drive circuit, it would be simple to add a high-decibel horn which sounds whenever a temperature reading goes above or below a specified level.
- **Increased Log Size.** The application can write to unused portions of the program EEPROM in the same manner as the data EEPROM. If the application is kept small enough, part of the program EEPROM can be used to store additional temperature log samples, thereby allowing a longer period of time to be covered.
- **Two-Way Serial Communications.** Implementing a two-way bit-banged serial port, while more complex than a transmit-only version, is well within the capabilities of the MAXQ3210. This process would allow a host to request a log upload from the MAXQ3210, set configuration values such as the temperature

resolution of the DS1822, query specific sensors on demand, and even potentially upload new firmware to the MAXQ3210 over the serial network.

## Conclusion

The MAXQ3210's compact size, low power consumption, and I/O flexibility make it an ideal choice for battery-powered environmental measuring applications. Several 1-Wire sensors are available to measure a variety of environmental conditions including temperature and humidity. These sensors can be easily interfaced to the MAXQ3210 with as little as a single port pin. Finally, the data can be stored for later retrieval and analysis, all using the nonvolatile EEPROM memory on the MAXQ3210 itself.

---

Application Note 3769: [www.maxim-ic.com/an3769](http://www.maxim-ic.com/an3769)

### More Information

For technical support: [www.maxim-ic.com/support](http://www.maxim-ic.com/support)

For samples: [www.maxim-ic.com/samples](http://www.maxim-ic.com/samples)

Other questions and comments: [www.maxim-ic.com/contact](http://www.maxim-ic.com/contact)

---

### Automatic Updates

Would you like to be automatically notified when new application notes are published in your areas of interest? [Sign up for EE-Mail™](#).

---

### Related Parts

DS1822: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

MAXQ3210: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN3769, AN 3769, APP3769, Appnote3769, Appnote 3769

Copyright © by Maxim Integrated Products

Additional legal notices: [www.maxim-ic.com/legal](http://www.maxim-ic.com/legal)