Keywords: RTC, 32kHz, real time clock, iButton, 1-Wire, one wire, clock

APPLICATION NOTE 4641

Real-Time Clock Adds Accurate Timekeeping to Microcontroller System

Apr 02, 2012

Abstract: The MAXQ610 microcontroller does not include a battery-backed real-time clock (RTC), but the flexibility of the 1-Wire® network makes it straightforward to add a DS1904 RTC iButton® to any MAXQ610-based application. Communicating with the DS1904, setting the clock and control values, and converting time values to and from a raw seconds count are all well within the capabilities of the MAXQ610, even when using assembly language. This application note explains how to add RTC functionality to a MAXQ610-based application. Demonstration code for implementing this application is shown. The principles and techniques in this article apply equally well to any other MAXQ®-based microcontroller that has general-purpose I/O (GPIO) pins capable of driving the 1-Wire communications protocol.

A similar version of this article appeared in the March 1, 2012 issue of *Chip Design* magazine.

Introduction

Many microcontrollers include timer circuits, but only a few include a battery-backed real-time clock (RTC). Yet many applications require an RTC, which you can easily add by using a 1-Wire network. This article explains how to add an RTC that supports the 1-Wire protocol to a microcontroller-based system. The necessary code is included. It explains principles and techniques that apply equally well to any microcontroller whose general-purpose I/O (GPIO) pins are capable of driving the 1-Wire communications protocol.

Design Goals

This demonstration shows how to implement methods for performing the following operations using the 1-Wire interface:

- Read the 64-bit ROM ID of the selected RTC
- Start and stop the RTC
- Read the current value of the RTC
- Set the RTC to a new value

The demo will also display the current RTC value in a readable format, i.e., convert from raw seconds to a year/month/day/time format. It will allow the user to modify the clock value by incrementing the various converted values (e.g., year, month, day) rather than calculating and entering a new value for the seconds.

As for any application that stores a count of seconds for the date/time value, we must choose a zero baseline. For this application, that baseline is January 1, 2000, at 12:00:00 AM, for which the raw seconds count is zero (00000000h).

System Setup

The 1-Wire interface is fundamental to this article. It lets you add an RTC that supports the 1-Wire protocol to any microcontroller. The DS1904 RTC iButton® will be used in this example. This application uses the MAXQ610 microcontroller because it can easily communicate with an RTC, set clock and control values, and convert between raw seconds and the corresponding calendar date, even when using assembly language.

The low-power MAXQ610 is well-suited for portable applications, but it lacks a battery-backed RTC. You can, however, connect this microcontroller to a dedicated RTC by using one of its GPIO pins. Demonstration code for the microcontroller has been written using the assembly-based MAX-IDE environment. It is designed to run on the Maxim evaluation (EV) kit, MAXQ610-KIT. Source code, project files, and additional documentation are all available for download.

Running the Application

You need the following hardware to run the demonstration code:

- MAXQ610-KIT EV kit
- 5VDC power supply
- Serial-to-JTAG or USB-to-JTAG interface board
- JTAG programming cable (2 × 5 ribbon cable with 0.100in. pin connectors)
- Straight-through DB9 serial-interface cable
- PC with available COM port or USB-to-serial adapter
- DS1904L-F5# RTC iButton
- DS9094F+ through-hole-mount iButton clip

The code runs on the MAXQ610 EV kit. An iButton clip (DS9094F+) is installed in the prototyping area and a DS1904L-F5# RTC iButton inserted in the iButton clip. Connections are then made from the iButton clip:

- Connect the GROUND pin of the iButton clip (the pin labeled "+" on the top side of the clip that contacts the back/unlabeled side of the DS1904) to one of the GND test points on the MAXQ610 EV kit.
- Connect the DATA pin of the iButton clip (the pin on the inside of the clip that contacts the front/labeled side of the DS1904) to port pin P2.0 (header pin P3.1) on the MAXQ610 EV kit.

You also need the following software:

- MAX-IDE assembly language development environment for a MAXQ microcontroller
- Microcontroller Tool Kit (MTK) or other terminal emulator with "dumb terminal" mode

The latest installation package and documentation for the MAX-IDE environment are available at our MAXQ RISC Microcontrollers page.

Data for the RTC is transferred serially over the 1-Wire protocol; only a single data lead and a ground return are required. This RTC contains a unique 64-bit ID, factory-lasered in ROM and an RTC/calendar implemented as a binary counter. It resides in a durable MicroCan package that resists dirt, moisture, and shock. This package can be mounted on almost any surface, including printed circuit boards (PCBs) and plastic key fobs. When operating, the RTC adds a calendar date, time and date stamp, stopwatch, hour meter, interval timer, and logbook function to any electronic device or embedded application that uses a microcontroller.

The RTC contains a 32-bit counter with 1-second resolution, which provides a range of approximately 136 years. All the hardware needed to keep the clock running, including the 32kHz crystal and a battery, are sealed inside. The resulting device has an operating life greater than 10 years, with clock accuracy of approximately ±2 minutes per month at a room temperature of +25°C. Operating mode (halted or running) and the value of the clock counter can be read or written using the 1-Wire interface.

Driving the 1-Wire Network

The 1-Wire interface provides power and communications over a single wire plus a single ground return. This means that a single port pin enables a microcontroller to communicate with a 1-Wire sensor. A variety of sensors and other components have been developed for operation on 1-Wire networks.

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

The 1-Wire line operates in an open-drain mode: the master (and also the slaves, when their output is requested) indicates "zero" by pulling the line to ground, or "one" by letting it float high. This operation is normally implemented with a discrete pullup resistor attached between the line and V_{CC}. Microcontrollers with a weak pullup mode on their port pins (like the MAXQ610) can simply switch the port pin back to that mode and let the line float high; no external resistor is required. Because the master and slaves pull the line low and never actively pull it high, the 1-Wire network operates in a wired-OR configuration. This approach prevents line conflict when multiple slaves attempt to transmit on the 1-Wire bus simultaneously.

To drive the 1-Wire network, the microcontroller uses software to generate time slots on a single pin. All time slots are initiated by the 1-Wire master, so the microcontroller does not need to monitor the 1-Wire line when it is not communicating with a slave device.[1]

- **Reset** time slots are approximately 1ms wide. For the first half of the time slot, the master (MAXQ610) holds the 1-Wire line low. Halfway through the time slot, it 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. The slaves then generate a **presence pulse**, which indicates to the master that one or more slaves are present and ready to communicate.

- **Write** time slots are 60µs to 120µs wide, and used by the master to transmit bits (0 or 1) to one or more slaves. Both types of write time slots start with the master pulling the line low for at least one microsecond. To transmit a 1, the master then releases the line (lets it float high) for the rest of the time slot. To transmit a 0, the master holds the line low until the end of the time slot.
- **Read** time slots are 60µs to 120µs wide, and used by the master to read bits (0 or 1) from a slave device. The time slot starts with the master pulling the line low for at least one microsecond. The master then releases the line, allowing the slave to either hold it low (0) or let it float high (1). Midway through the time slot, the master samples the line to read the bit value from the slave.

The MAXQ610 runs at about 12 instruction cycles per microsecond at 12MHz, so it easily executes the standard 1-Wire protocol in software using a port pin (P2.0). It implements read time slots in a similar manner. Note that all data bytes on the 1-Wire bus are transmitted least significant bit (LSB) first.

The value of the pullup resistor on the 1-Wire bus varies according to the number of devices on the network, but is typically specified at 4kΩ to 5kΩ. In contrast, the weak pullup resistor on the MAXQ610 port pin varies from 15kΩ to 40kΩ, depending on the operating voltage. To avoid an excessive time interval on the 1-Wire bus as it floats high from the low state, the code briefly drives the bus (via P2.0) with a normal high state that "snaps" the bus to the high state before setting P2.0 to the normal weak-pullup mode. This action causes no disruption on the 1-Wire bus if, that is, you avoid those time intervals in which the slave might attempt to pull the bus low. As an alternative, you can put a physical external pullup resistor on the 1-Wire bus, and then drive the port pin in standard low mode for a zero state and to the tristate mode for a high state.

Starting, Stopping, and Setting the Clock

Because more than one 1-Wire device may be present on the 1-Wire bus, communication with these devices proceeds in two stages. The bus master first selects a 1-Wire device with which to communicate, and then issues the communication.[2] After the bus master transmits a reset pulse, all slave devices on the 1-Wire bus return to the default unselected state. Several commands are then available to the bus master for selecting the device with which it will communicate in the second stage. The following commands use the 64-bit ROM ID associated with each slave device. The commands are supported by all 1-Wire devices.

Skip ROM [CCh]

This single-byte command activates all slave devices on the bus. It is useful if only a single 1-Wire device is present, or if the bus master needs to send the same command to all 1-Wire devices on the bus. The application above has only one device on the bus (e.g., the DS1904 RTC), so the bus master (e.g., the MAXQ610 microcontroller) uses this command throughout to activate the RTC before reading from or writing to it.

Read ROM [33h]

This single-byte command activates all slave devices on the bus and causes them to transmit their 64-bit ROM ID values back to the bus master. Since it activates all slave devices, it can only be used for single-slave systems. Otherwise, multiple slave devices will cause data collisions as they attempt to transmit their ROM IDs simultaneously. Because only one device (DS1904) is present on the bus in our application, the MAXQ610 uses this command at the beginning to read the ROM ID of the DS1904.

Match ROM [55h]

This command selects one slave among multiple slaves on the 1-Wire bus. After the bus master transmits this command, it follows up by transmitting the 64-bit ROM ID of the slave device to be selected. The device with matching ROM ID responds by going to an active state, while all other devices on the bus go inactive and await the next 1-Wire reset from the bus master. (This command is not used in the application described here.)

Search ROM [F0h]

This command lets the bus master use an iterative discovery process to determine the ROM ID values of one or more slave devices on the 1-Wire bus.[3] (This command is not used in the application described here.)

Reading and Writing the Clock and Control values

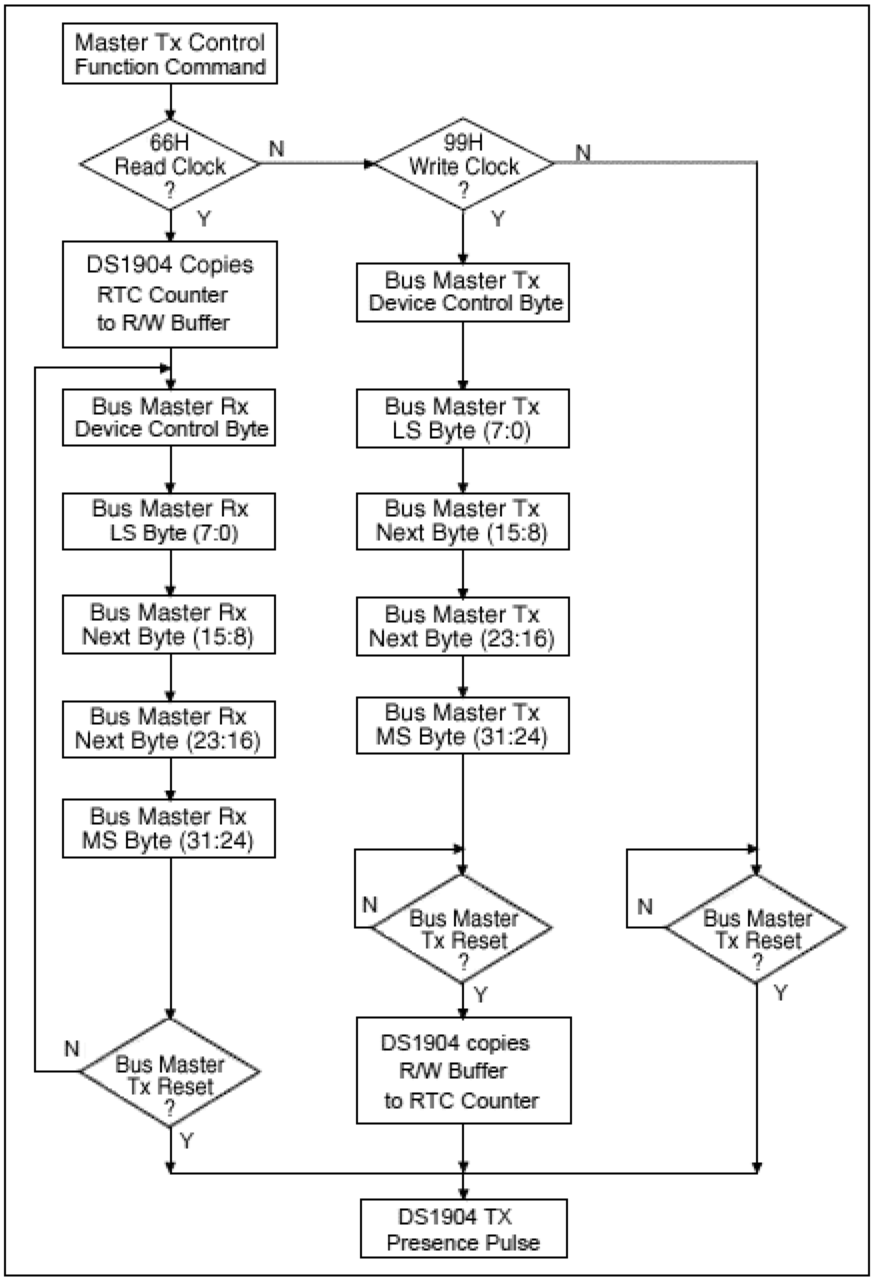
When the bus master has selected the 1-Wire slave device (i.e., the RTC, DS1904) using the Skip ROM or Read ROM command, that device is ready to accept 1-Wire commands specific to it. These commands (**Figure 1**) are detailed below:

Read Clock [66h]

This command allows the bus master to read both the device control byte from the DS1904 and the 4-byte (32-bit) RTC value. The device control byte determines whether the 32kHz oscillator that drives the RTC is running or stopped. As shown by the code below, only one command reads both the device control byte and the clock value. Even if both values are not needed, you must read the device control byte before the device will output the clock data.

Write Clock [99h]

As the complement to Read Clock, this command allows the bus master to set new values for the device control byte and the DS1904 4-byte clock counter. Note that you must write all 5 bytes and transmit a 1-Wire reset pulse before the new values take effect. The application code above includes routines that set the device control byte and clock value individually, by first reading the 5 bytes of data from the DS1904 (1 byte of device control plus 4 bytes of the clock counter), and then writing back the data with the appropriate changes.



Master Tx Control Function Command
66H Read Clock ?
N
Y
99H Write Clock ?
N
Y
DS1904 Copies RTC Counter to R/W Buffer
Bus Master Rx Device Control Byte
Bus Master Rx LS Byte (7:0)
Bus Master Rx Next Byte (15:8)
Bus Master Rx Next Byte (23:16)
Bus Master Rx MS Byte (31:24)
Bus Master Tx Device Control Byte
Bus Master Tx LS Byte (7:0)
Bus Master Tx Next Byte (15:8)
Bus Master Tx Next Byte (23:16)
Bus Master Tx MS Byte (31:24)
Bus Master Tx Reset ?
DS1904 copies R/W Buffer to RTC Counter
DS1904 TX Presence Pulse

Figure 1. These DS1904 clock-function commands are taken from the data sheet.

Converting Time and Date Values

To convert the raw seconds count into printable form, the application determines the value of each date and time field (year, month, day, hour, minute, and second) individually, beginning with the largest field (the year) and working downward:

1. While Seconds ≥ (seconds per year), subtract (seconds per year) from Seconds and increment Year.
2. While Seconds ≥ (seconds per month), subtract (seconds per month) from Seconds and increment

Month.

3. While Seconds ≥ (seconds per day), subtract (seconds per day) from Seconds and increment Day.
4. While Seconds ≥ (seconds per hour), subtract (seconds per hour) from Seconds and increment Hour.
5. While Seconds ≥ 60, subtract 60 from Seconds and increment Minute.
6. The remaining value of seconds is the Second field.

Even if the bus master provides hardware support for division, a simple division operation is not enough to calculate the first two field values (years and months). This is because the number of seconds per field unit changes due to the effect of leap years (which affects the value for years and months) and the number of days per month (which affects months only). As an example, start with the year 2000 (a leap year):

- Seconds per year in 2000) = 366 (days) × 24 (hours/day) × 60 (min/hour) × 60 (sec/min) = 31,622,400 seconds.
- Standard years have one less day (365 days), which changes the seconds/year to (31,622,400 - 86,400) = 31,536,000.

Because every fourth year is a leap year, we calculate Year as follows (note that items 2, 3, and 4 in this pseudocode are identical.):

1. If Seconds ≥ (seconds per leap year), subtract (seconds per leap year) from Seconds and increment Year, otherwise stop.
2. If Seconds ≥ (seconds per year), subtract (seconds per year) from Seconds and increment Year, otherwise stop.
3. If Seconds ≥ (seconds per year), subtract (seconds per year) from Seconds and increment Year, otherwise stop.
4. If Seconds ≥ (seconds per year), subtract (seconds per year) from Seconds and increment Year, otherwise stop.
5. Go back to line 1.

The value of the Month field is calculated in a similar manner:

1. If Seconds ≥ (seconds in January), subtract (seconds in January) from Seconds and increment Month, otherwise stop.
2. If Seconds ≥ (seconds in February), subtract (seconds in February) from Seconds and increment Month, otherwise stop.
3. If Seconds ≥ (seconds in March), subtract (seconds in March) from Seconds and increment Month, otherwise stop.
4. Proceed through the remaining months.

Running the Demo

To run the demo, load and run the application. Then use the DB9 serial cable to connect the MAXQ610 EV kit from J1 SKT to COM1 on the PC. Start MTK (or another terminal emulator) and open COM1 at 38400 baud. The initial output should be similar to the following:

```
@
```

```
ID: 24B91231000000B2  AC  18F83065

+   18F83065   Apr 10 2013, 02:15:01 pm
+   18F83066   Apr 10 2013, 02:15:02 pm
+   18F83067   Apr 10 2013, 02:15:03 pm
+   18F83068   Apr 10 2013, 02:15:04 pm
+   18F83069   Apr 10 2013, 02:15:05 pm
```

The second line of code contains the DS1904 ROM ID value (24B91231000000B2), the device control byte (AC), and the current clock value (18F83065). The "+" value in subsequent lines indicates that the clock is running. The time value is refreshed and displayed as often as it changes, which should be once per second. Press "-" to stop the clock. At that point you can alter the clock values by pressing additional keys:

- + Start the clock and begin auto-updating again.
- Y Increment the year value, and reset month and day to 01/01.
- M Increment the month value, and reset the day to 01.
- D Increment the day value (wraps around, depending on the current month).
- h Increment the hour value.
- m Increment the minute value.
- s Reset the seconds counter to 00.
- Z Zero the seconds counter, resetting the time to Jan 01 2001, 12:00:00 am.

References

1. Refer to the DS1904 RTC data sheet for more details on 1-Wire timing requirements.
2. For more details on the following commands, please refer to the DS1904 data sheet.
3. Ibid., www.maximintegrated.com/DS1904.

Related Parts		
DS1904	RTC iButton	Free Samples
MAXQ610	16-Bit Microcontroller with Infrared Module	Free Samples
MAXQ610-KIT	Evaluation Kit for the MAXQ610	

More Information
For Technical Support: http://www.maximintegrated.com/support