Keywords: real-time clock, BCD, RTC, binary coded decimal, state machine logic

TUTORIAL 5413

State Machine Logic in Binary-Coded Decimal (BCD)-Formatted Real-Time Clocks

Sep 17, 2012

Abstract: When developing code to operate a real-time clock (RTC), it is often beneficial to understand how the clock core operating logic has been defined. This tutorial discusses the counter-chain structure used in Maxim's binary-coded decimal (BCD)-formatted RTCs. It provides some insight into the expected chip behavior if improper or illegal contents are somehow installed.

A similar version of this article appears on *Electronic Design*, July 2012.

Introduction

When developing code to operate a real-time clock (RTC), it is often beneficial to understand how the clock core operating logic has been defined. We all have a fundamental comprehension of how a clock and calendar should operate when properly programmed, but this application note also provides some insight into the expected chip behavior if improper or illegal contents are somehow installed.

A basic real-time clock counter chain found in many of Maxim's binary-coded decimal (BCD)-formatted RTCs (such as the DS12885 and DS1302) is illustrated in **Figure 1**.

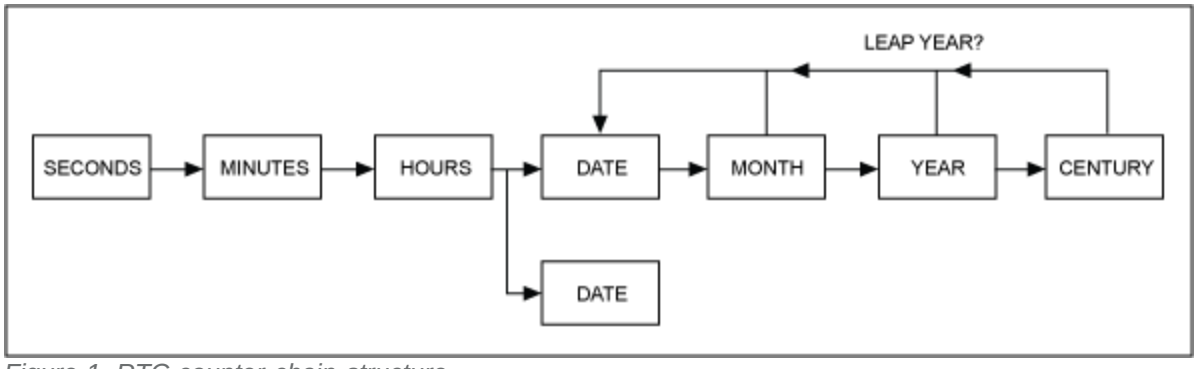

Figure 1. RTC counter chain structure.

Each counter register has defined minimum and maximum values, and most are preset to their appropriate minimums on initial power application. As a natural function of timekeeping, when the Seconds count reaches the maximum value, the next increment causes a carry to the Minutes register and the Seconds rolls over to the minimum value. A Minutes rollover carries to the Hours; the Hours rollover carries to both the Date and arbitrary Day-of-Week registers; the Date rollover carries to the

Month register; the Month rollover carries to the Year register; and if applicable, the Year rollover carries to the Century register. **Table 1** lists the registers and associated range of legal values.

Table 1. RTC Register Range and Carry Function

Register	Minimum (hex)	Maximum (hex)	Carry to Register
Hundredths	00	99	Seconds
Seconds	00	59	Minutes
Minutes	00	59	Hours
Hours (12-hour mode)	41	72	AM ⟶ PM, PM ⟶ AM + Day & Date
Hours (24-hour mode)	00	23	Day & Date
Day	01	07	—
Date	01	31*	Month (*Month and Year dependent)
Month	01	12	Year
Year	00	99	Century (register or bit)
Century	00	99	—

Some variations on the counter implementation may include a 'Hundredths of a Second' register (precedes Seconds), and a Century bit to flag the Year register rollover.

Register Descriptions

For each register, the normal state machine logic is to increment the BCD count from the minimum to the defined maximum, and then roll back over to the minimum while applying the carry to the next register. For an example, the Date register BCD count sequence for a 30-day month would be 01h...09h, 10h...19h, 20h...29h, 30h, and then roll back to 01h.

Maxim does not critique the values a user might install during operation, so the ramifications of loading improper values are based upon the specific component and which register(s) was actually involved. ***Illogical time and date entries may result in unexpected operation.***

The **Hundredths of a Second** register (if applicable) counts from 00h to 99h. The next increment causes the counter to roll over to 00h and the carry is applied to the Seconds register. All 8 bits in this register are read/write capable. Devices containing this register are internally clocked from the 4kHz time base (32.768kHz/8).

The ***Seconds*** register counts from 00h to 59h. The next increment causes the counter to roll over to 00h and the carry is applied to the Minutes register. Only the least significant 7 bits in this register are usually read/write capable. Most devices *not containing* a Hundredths of a Second register are internally clocked

from the 1Hz time base.

The **Minutes** register also counts from 00h to 59h. The next increment causes the counter to roll over to 00h and the carry is applied to the Hours register. Only the least significant 7 bits in this register are read/write capable.

The **Hours** register accommodates either a 12-hour format (with an AM/PM designator) or a 24-hour clock. A control bit (usually bit6 = 12/active-low 24) is used to identify the clock operating mode. Only the least significant 7 bits in the Hours register are read/write capable. In the 24hour Mode (bit6 = 0), the Hours register counts from 00h to 23h. The next increment causes the counter to roll-over to 00h and the carry is applied to both the Date and Day registers.

In the 12-hour mode (bit6 = 1), the Hours register count sequence, starting at "12AM", counts 52h, 41h...49h, 50h, and then 51h. The next increment causes the counter to roll over to 72h with the carry applied to the active-low AM/PM bit (PM = 1). The register then counts 72h, 61h...69h, 70h, 71h and the next increment causes the counter to roll over to 52h as PM is cleared, with the carry applied to both the Date and Day registers. For clarification, 12 midnight is 12:00:00 AM and 12 noon is 12:00:00 PM.

The **Day** (Day-of-Week) register is the most simplistic counter, with an intended content of 01h to 07h. The next increment causes the counter to roll over to 01h with no carry. Should a user accidentally deposit a 00h content into this register, there is no adverse effect upon the other timekeeping registers, but the first week of clock operation will have 8 days (0 → 7). The Day register only has three functioning bits, so even though it initially seems impossible for the user to load a value greater than 7, any hex value > 7 written would store the *bad value* logic-ORd with the three functioning bits (07h) (e.g., writing 1Fh would store a 07h, or writing F5h would store a 05h). Normal calendar convention is to define Sunday= 1, but that definition is arbitrary and fully under the user's control.

The **Date** register counts from 01h to the defined *end of the present Month*. The *end of the month* is dependent upon the present Month and Year content (in the case of February). Only the least-significant 6 bits in the Date register are read/write capable. **Table 2** lists the three options.

Table 2. Last Date of Month

Calendar Month	Numeric Month	End Date
January, March, May, July, August, October, December	1, 3, 5, 7, 8, 10, 12	31
April, June, September, November	4, 6, 9, 11	30
February	2	28 (typical year), 29 (leap year)

The **Month** register counts from 01h to 12h. The next increment causes the counter to roll over to 01h and the carry is applied to the Year register. To address Y2K issues in the late 20th Century, a Century bit was included in the MSB of the Month register on components that did not include a full 8-bit Century register. This Century bit facilitated easy detection of the century change, as the bit changes state upon a Year register rollover from 99h to 00h. Care should be taken when loading new calendar values to avoid corrupting the Century bit logic condition. Please refer to the register map for the specific component in question; conventional register placement leaves bits 7 and 4:0 as read/write capable.

The **Year** register counts from 00h to 99h. The next increment causes the counter to roll over to 00h and the carry is applied to either the Century register or the Century bit, whichever is applicable to the component in question. All 8 bits in this register are read/write capable.

Leap Year compensation is based upon the era of the specific chip design, and applicability is normally annotated on the cover page of that data sheet. For most chip designs that do not include a Century register, the Leap Year compensation functions properly through 2099 and is assumed to be the result of this expression:

```
If MODULO (CALENDAR_YEAR/4) == 0
```

Due to the era of existing chip designs, many of today's RTC components containing a Century register apply the same algorithm solely upon on content of the Year register, and therefore will incorrectly designate 2100 as a leap year.

The DS1347 RTC utilizes the full Gregorian algorithm to determine Leap Year adjustments for February:

```
        Leap_Year = 0                                                    ;
default to No
        If MODULO (CALENDAR_YEAR/4) == 0                                 ;
divisible by 4?
                If MODULO (CALENDAR_YEAR/100) == 0                       ;
divisible by 100?
                        If MODULO (CALENDAR_YEAR/400) == 0               ;
divisible by 400?
                                Leap_Year = 1
                        end                         ; if
                else
                                Leap_Year = 1
                end                                 ; if
        end                                         ; if
```

The **Century** register (if applicable) counts from 00h to 99h. All 8 bits in this register are read/write capable.

Results When Illegal Data Is Written

Up to this point, we have discussed the proper counting methods used by the RTC, given the legal range of values for the register in question. Occasionally a question may arise concerning expected counter behavior when improper values were initially installed. The answer is deterministic, and only requires a copy of the component's register map to arrive at a conclusion.

To access the result of loading an improper value, perform a Logic AND of the written value with the write-enabled bits in the register in question.

If the resulting content is *less than the register's maximum counter value*, the register should subsequently increment from that AND-ed value. Rollover will occur when the bit states in the register eventually coincide with the expected maximum value for that counter (12h or 00010010b as shown in **Example 1**).

Example 1: Write 2Ah into the Month register of a DS1337 serial RTC

From the product specification, the Month register content was verified to match the register description above.

BCD	Binary	
2Ah	00101010	Write to Month
9Fh	10011111	Enabled bits in Month register (with Century bit)
0Ah	00001010	Resulting content of "0Ah" after writing the Month
12h	00010010	Logic match for "maximum value"

If the resulting content is *greater than the register's maximum counter value*, the register should subsequently increment from that AND-ed value to some unpredictable rollover condition. In this case, not only was the hour value illogical, but setting bit6 also changed the functionality of the Hours counter to 12-hour Mode. Prediction of the roll-over point is illogical and results may vary, based upon that specific chip design.

Example 2: Write FFh into the Hours register of a DS1390 low-voltage RTC

From the product specification, the Hours register content was verified to match the register description above.

BCD	Binary	
FFh	11111111	Write to Hours
7Fh	01111111	Enabled bits in Hours register
7Fh	01111111	Resulting content of "7Fh" when reading the Hours

Example 3: Write 38h into the Hours register of a DS1341 low-current RTC

From the product specification, the Hours register content was verified to match the register description above.

Since bit6 (12/active-low 24) was *not set*, the component would be running in 24-hour mode, and 38h is obviously past the normal 24-hour mode register maximum of 23h. The resulting time error is difficult to estimate, given an unstated period of operation between when the value was corrupted and when it was detected.

BCD	Binary	
38h	00111000	Write to Hours
7Fh	01111111	Enabled bits in Hours register
38h	00111000	Resulting content of "38h" when reading the Hours

Example 4: Write AAh into the Seconds register of a DS1302 trickle-charge timekeeping chip

From the product specification, the Seconds register content deviates from the register description above; the inclusion of the CH bit (bit7) requires some further understanding of that bit's functionality.

Since only 7 bits are used for the Seconds value, the counter was actually written to 2Ah. "CH" stands for *Clock Halt*.

With CH = 1, the oscillator is stopped and the counters do not increment, so it will never increment or roll over with CH set. The resulting time loss is difficult to estimate, given an unstated period of this idle (noncounting) condition before detection/correction.

BCD	Binary	
AAh	10101010	Write to Seconds
FFh	11111111	Enabled bits in Seconds register
AAh	10101010	Resulting content of "AAh" when reading the Seconds

Example 5: Write 2Dh into the Date register of a DS1338 serial RTC

From the product specification, the Date register content was verified to match the register description above.

BCD	Binary	
2Dh	00101101	Write to Date
3Fh	00111111	Enabled bits in Hours register
2Dh	00101101	Resulting content of "2Dh" when reading the Hours

Summary

In conclusion, we have discussed the counter chain structure utilized in Maxim's BCD-formatted RTCs, the derivation of unique calendar events, and Leap Year/Century handling. Additionally, using published register information and the deterministic nature of the state machine logic, some behavioral traits resulting from errant programming can be recognized during the system debugging phase. This symptom recognition can reduce a product's time to market.

Related Parts		
DS1302	Trickle-Charge Timekeeping Chip	Free Samples
DS1305	Serial Alarm Real-Time Clock	Free Samples
DS1306	Serial Alarm Real-Time Clock	Free Samples
DS1307	64 x 8, Serial, I²C Real-Time Clock	Free Samples

DS1337	I²C Serial Real-Time Clock	Free Samples
DS1338	I²C RTC with 56-Byte NV RAM	Free Samples
DS1339	I²C Serial Real-Time Clock	Free Samples
DS1340	I²C RTC with Trickle Charger	Free Samples
DS1341	Low-Current I²C RTCs for High-ESR Crystals	Free Samples
DS1342	Low-Current I²C RTCs for High-ESR Crystals	Free Samples
DS1343	Low-Current SPI/3-Wire RTCs	Free Samples
DS1344	Low-Current SPI/3-Wire RTCs	Free Samples
DS1347	Low-Current, SPI-Compatible Real-Time Clock	Free Samples
DS1375	I²C Digital Input RTC with Alarm	Free Samples
DS1388	I²C RTC/Supervisor with Trickle Charger and 512 Bytes EEPROM	Free Samples
DS1390	Low-Voltage SPI/3-Wire RTCs with Trickle Charger	Free Samples
DS1391	Low-Voltage SPI/3-Wire RTCs with Trickle Charger	Free Samples
DS1392	Low-Voltage SPI/3-Wire RTCs with Trickle Charger	Free Samples
DS1393	Low-Voltage SPI/3-Wire RTCs with Trickle Charger	Free Samples
DS1394	Low-Voltage SPI/3-Wire RTCs with Trickle Charger	Free Samples
DS1629	Digital Thermometer and Real-Time Clock/Calendar	Free Samples
DS1670	Portable System Controller	Free Samples
DS1673	Portable System Controller	Free Samples
DS1678	Real-Time Event Recorder	
DS3231	Extremely Accurate I²C-Integrated RTC/TCXO/Crystal	Free Samples
DS3231M	±5ppm, I²C Real-Time Clock	Free Samples
DS3232	Extremely Accurate I²C RTC with Integrated Crystal and SRAM	Free Samples
DS3232M	±5ppm, I²C Real-Time Clock with SRAM	Free Samples
DS3234	Extremely Accurate SPI Bus RTC with Integrated Crystal and SRAM	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact