Application Note 517
DS1371/DS1374 32-Bit Binary Counter Time Conversion

www.maxim-ic.com

INTRODUCTION

The DS1371 and DS1374 have a 32-bit binary counter RTC with a 2-wire interface. The 32-bit binary counters are designed to continuously count time in seconds. This application note addresses how to convert the 32-bit time value into a date and time value that can be put in the form of MM/DD/YYYY, HH:MM:SS. An algorithm for converting from a date and time to binary seconds is also described.

Both devices accumulate time information in a 4-byte register as the number of seconds since some arbitrary reference date. The date used in this application note is the same used with the UNIX operating system, the reference date January 1, 1970, often referred to as Unix Epoch. **Note:** Because of the reference date used and the use of a 32-bit counter, this algorithm rolls over on Tuesday, January 19, 03:14:07, 2038.

BINARY SECONDS TO DATE/TIME

Figure 1 shows the basic algorithm used to convert raw seconds to a date/time. Below is a C implementation of the algorithm.

```
void DS1371_BinaryToDate(unsigned long binary,tm_struct *datetime) {

      unsigned long hour;
      unsigned long day;
      unsigned long minute;
      unsigned long second;
      unsigned long month;
      unsigned long year;

      unsigned long whole_minutes;
      unsigned long whole_hours;
      unsigned long whole_days;
      unsigned long whole_days_since_1968;
      unsigned long leap_year_periods;
      unsigned long days_since_current_lyear;
      unsigned long whole_years;
      unsigned long days_since_first_of_year;
      unsigned long days_to_month;
      unsigned long day_of_week;

      whole_minutes = binary  / 60;
      second = binary - (60 * whole_minutes);              // leftover seconds

      whole_hours  = whole_minutes  / 60;
      minute = whole_minutes - (60 * whole_hours);         // leftover minutes

      whole_days   = whole_hours  / 24;
      hour         = whole_hours - (24 * whole_days);      // leftover hours

      whole_days_since_1968 = whole_days + 365 + 366;
      leap_year_periods = whole_days_since_1968 / ((4 * 365) + 1);

      days_since_current_lyear = whole_days_since_1968 % ((4 * 365) + 1);
```


REV: 082103

```
	// if days are after a current leap year then add a leap year period
	if ((days_since_current_lyear >= (31 + 29))) {
		leap_year_periods++;
	}
	whole_years = (whole_days_since_1968 - leap_year_periods)  / 365;
	days_since_first_of_year = whole_days_since_1968
					- (whole_years * 365) - leap_year_periods;

	if ((days_since_current_lyear <= 365) && (days_since_current_lyear >= 60)) {
		days_since_first_of_year++;
	}
	year = whole_years + 68;

	// setup for a search for what month it is based on how many days have past
	//   within the current year
	month = 13;
	days_to_month = 366;
	while (days_since_first_of_year < days_to_month) {
		month--;
		days_to_month = DaysToMonth[month];
		if ((month >= 2) && ((year % 4) == 0)) {
			days_to_month++;
		}
	}
	day = days_since_first_of_year - days_to_month + 1;

	day_of_week = (whole_days  + 4) % 7;

	datetime->tm_yday =
		days_since_first_of_year;	/* days since January 1 - [0,365]   */
	datetime->tm_sec  = second;		/* seconds after the minute - [0,59]*/
	datetime->tm_min  = minute;		/* minutes after the hour - [0,59]  */
	datetime->tm_hour = hour;		/* hours since midnight - [0,23]    */
	datetime->tm_mday = day;		/* day of the month - [1,31]        */
	datetime->tm_wday = day_of_week;	/* days since Sunday - [0,6]        */
	datetime->tm_mon  = month;		/* months since January - [0,11]    */
	datetime->tm_year = year;		/* years since 1900                 */
}
```

DATE/TIME TO BINARY SECONDS

Figure 2 shows the basic algorithm used to convert date/time to raw seconds. Below is a C implementation of the algorithm.

```
unsigned long DS1371_DateToBinary(tm_struct *datetime) {

	unsigned long iday;
	unsigned long val;

	iday = 365 * (datetime->tm_year - 70) + DaysToMonth[datetime->tm_mon]
				+ (datetime->tm_mday - 1);
	iday = iday + (datetime->tm_year - 69) / 4;
	if ((datetime->tm_mon > 1) && ((datetime->tm_year % 4) == 0)) {
		iday++;
	}
	val = datetime->tm_sec + 60 * datetime->tm_min + 3600
				* (datetime->tm_hour + 24 * iday);
	return val;
}
```

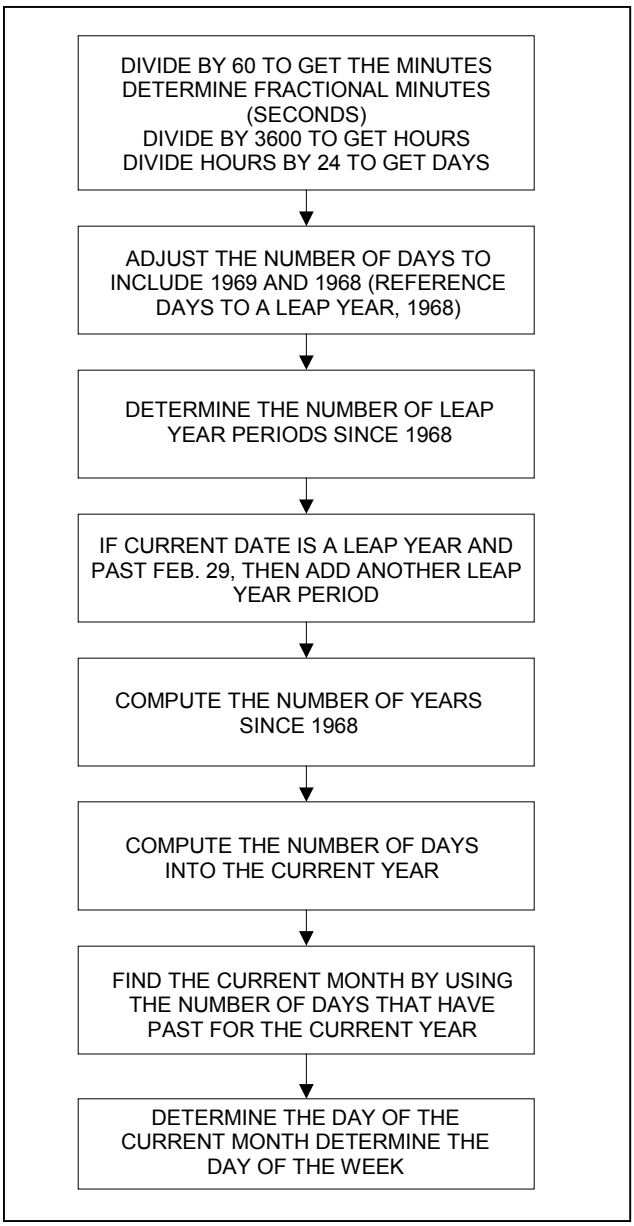

Figure 1. Binary to Date Algorithm Flow

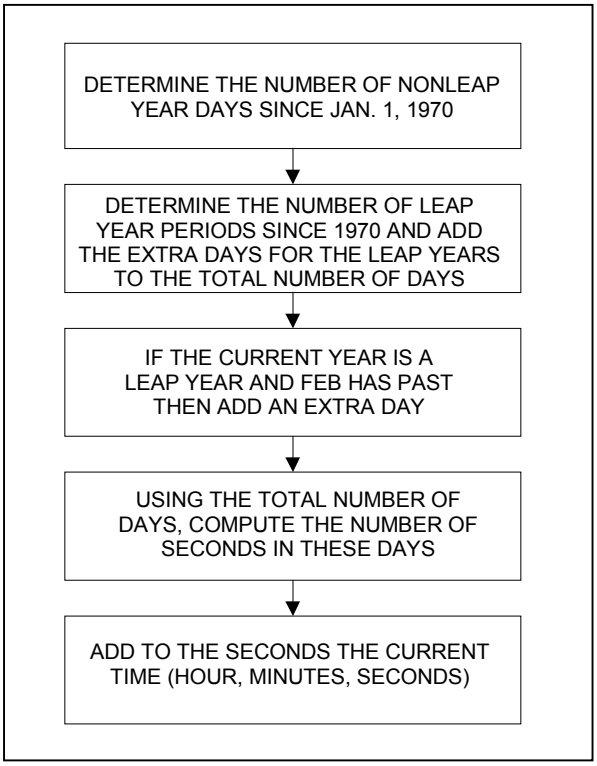

Figure 2. Date to Binary Algorithm Flow

TESTING

The C implementations used in this application note were thoroughly tested for each counter value over the 68-year span of the algorithm (from January 1, 1970 to January 18, 2038) and was found to be error free.

SOFTWARE

The software C routines of the algorithm as well as the software used to test the algorithms can be found on our ftp site: ftp://ftp.dalsemi.com/pub/timekeeping/DS1371/.