Low-cost Real-time Decoding of Broadcast Time and Frequency Standard

Amey Deosthali and Srikanth Gummadi
Department of Electrical and Computer Engineering
The University of Texas, Austin, TX 78712-1084
E-mail: {amey,gummadi}@vision.ece.utexas.edu

May 8, 1998

Contents

Abstract

The Time and Frequency Division of the National Institute of Science and Technology (NIST), which is located in Boulder, Colorado, is responsible for maintaining and distributing the time and frequency standard for the United States. NIST broadcasts the time and frequency standard from radio stations WWV, WWVH, WWVB, and GOES (Geostationary Operational Environmental Satellite) satellites. We present a new approach to decode the WWVB broadcast time information and to generate an accurate frequency reference calibrated to the NIST primary standard. The key innovations are that we develop new, zero-buffering algorithms and implement them on a microcontroller to decode the WWVB time code information and to use a pulse-width modulator to generate an accurate frequency reference. The microcontroller-based decoder provides a convenient and automatic way of setting the time and a secondary frequency standard referenced to the NIST primary standard. Our decoder for decoding the broadcast time information and generating an accurate frequency reference is the first to be implemented on a microcontroller and is at least half the cost of the existing WWVB decoders.

1 Introduction

Many businesses such as electric power companies, radio and television stations, telephone companies, aerospace industry, and the computer network industry require precise frequency and global time information. These users need to calibrate their timing to a reliable and internationally recognized standard. The National Institute of Standards and Technology (NIST) provides this standard for most users in the United States using radio services such as WWV located in Ft. Collins, Colorado, WWVH located in Kauai, Hawaii, and WWVB located in Ft. Collins, Colorado, as well as GOES (Geostationary Operational Environmental Satellite) satellites [1].

Existing WWVB time decoders are available in the form of integrated circuits. The clocks using such integrated receivers range in price from $30 to $200. In this project, we focus on the design of a microcontroller-based radio-controlled clock which provides accurate time synchronization and frequency calibration by decoding the WWVB signal. The radio-controlled clock automatically sets itself to the radio signal that is transmitted by the WWVB station. Apart from providing accuracy, the radio-controlled clock provides a convenient and automatic way of setting the time shown by the clock to the correct time.

In our decoder we sample the 60 kHz analog signal to get a digital signal and then track the power of the digital signal to decode the time information. Our technique has low complexity and is ideal for an implementation on a microcontroller which range from $2 to $15 in volumes of 100. We implement the time information decoder and the frequency reference generator on PIC microcontrollers which cost about $6 in volumes of 100. Taking other peripheral costs into consideration (such as the antenna cost), the microcontroller based decoder should still be able to reduce the cost of a WWVB decoder by half as compared to the existing decoders. To the best of the knowledge of the authors, this is the first radio clock that has been implemented on a microcontroller.

The key innovations of our technique are new zero-buffering algorithms and microcontroller implementations to decode the WWVB time code information and the use of a pulse-width modulator to generate an accurate secondary frequency reference from the WWVB signal. The key to the microcontroller implementation is to use bandpass sampling of the WWVB signal to reduce the sampling rate by a factor of 20. Our decoder provides a reliable, low-cost alternative to the existing WWVB decoders.

In Section 2, we describe the time code transmitted by radio station WWVB and discuss

the Friedman Interpolator. Section 3 describes the modeling of the radio clock in Ptolemy. Section 4 focuses on the implementation of the radio clock on a microcontroller.

2 Background

2.1 WWVB Time Code

Radio station WWVB continuously broadcasts time and frequency signals at 60 kHz, primarily for the continental United States. It broadcasts at a rate of 1 pulse per second using pulse-width-modulation. Each pulse is generated by reducing the carrier power by 10 dB at the start of the second, so that the leading edge of every negative-going pulse is on time. Full power is restored either 0.2, 0.5 or 0.8 seconds later to convey either binary "0", binary "1" or a *reference marker*, respectively [2].

The WWVB time code is sent in Binary Coded Decimal (BCD) format [3]. Every minute, the WWVB time code sends the current minute, hour, day of year, 2 digits of the current year, and Daylight Saving Time (DST) and leap year indicators. The coded information refers to the time at the start of the one-minute frame. Seconds are determined by counting pulses within the frame. Each minute begins with a *reference marker* pulse lasting for 0.8 seconds. A position identifier pulse (*reference marker*) lasting for 0.8 seconds is transmitted every 10 seconds.

2.2 Friedman Interpolator

The Friedman interpolator is an algorithm for estimating the frequency of a single sinusoid in white noise, based on the computation of the interval between zero crossings [4]. In the following analysis, only the negative to positive going zero crossings of the sinusoid are considered.

At the arrival of the first positive sample following the zero crossing, the estimate of the period of the sinusoid $T_e(n)$ is computed as

$$T_e(n) \;=\; [K(n) - \delta(n) + \delta(n-1)]T_{\rm s} \tag{1}$$

where $K(n)$ is the number of sampling intervals between the samples following the $(n-1)^{th}$ and n^{th} zero-crossings, and $\delta(n)T_s$ and $\delta(n-1)T_s$ are the time intervals between these zero crossings and the next positive samples (Figure 1).

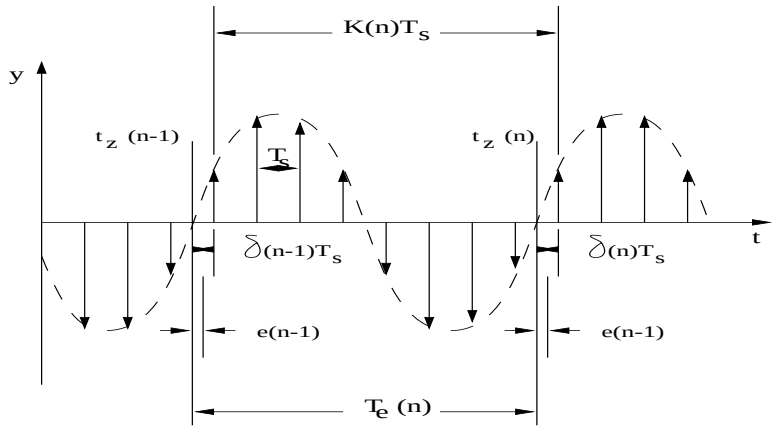

Figure 1: Frequency estimation by zero-crossing detection

If $e(n)$ is the error made in the computation of the n^{th} zero crossing, it can be shown [4] that

$$T_e(n) = T_{\sin} + e(n) - e(n-1) \tag{2}$$

where $T_{\sin}$ is the actual period of the sine wave. The spectrum of $T_e(n)$ contains a DC component which is equal to the actual period of the incoming signal, and the spectrum of the error signal which is concentrated in the high-frequency region. Thus, by using an appropriate low-pass filter, the period, and thus the frequency, can be computed to an arbitrary degree of accuracy (of the order of 10^{-4} to 10^{-6}).

We modified the above algorithm to estimate the phase information of the received signal. By knowing the phase of the sinusoid we can know the exact time that has elapsed from the previous sample. Thus we will be able to generate an accurate frequency reference.

3 Modeling the decoder

The decoder has two main parts: a signal processing front-end and a decision logic back-end. The signal processing front-end is modeled in the Synchronous Dataflow (SDF) domain. It has two main components: a sampler, which samples the continuous-time analog signal to convert it into a discrete-time digital signal, and a power estimator, which estimates the power of the received signal.

The decision logic back-end is modeled in the Finite State Machine (FSM) domain. It makes the decisions on the bits that are supplied by the signal processing front-end. The decision logic back-end is responsible for calculating the correct local time from the

transmitted Coordinated Universal Time (UTC).

3.1 Signal Processing Front-end

The decoder has an analog receiver that consists of a loop antenna for receiving the low-frequency band, and a high-Q amplifier for selecting and amplifying the 60 kHz signal. This analog signal is the input to the analog-to-digital converter.

3.1.1 Sampling the Analog Signal

In this application, the transmitted signal is a pure sinusoidal signal of 60 kHz and not a modulated signal. Sampling a pure sinusoid at a frequency which is lower than the Nyquist rate and following it by a low pass filter would result in another sinusoid but of a different frequency [5]. Since our aim is to track the changes in signal power, frequency of the signal is not important. The sampling frequency is restricted by the following factors :

1. The transmitted frequency must not be a multiple of the sampling frequency. Otherwise the low-pass component of the aliased signal would be the DC component. Thus all the information about the signal will be lost.
2. The sampling frequency should be low enough so that we can perform all the required calculations on the present sample before the next sample arrives.
3. The sampling frequency should be as high as possible so that we get more number of samples to estimate the power (hence more accurate).
4. The sampling frequency should map as an integer count on the microcontroller.
5. The sampling frequency should be greater than 2 Hz (twice bandwidth of message signal)

Taking all the above conditions into consideration we choose 6.25 kHz as the sampling frequency.

3.1.2 Estimating Signal Power

Accurate estimation of the power of the received signal is a very important task in the decoding of the WWVB signal. The signal power estimator has to be simple, efficient, accurate, and fast. In our implementation we have used a single pole infinite impulse response

(IIR) filter to estimate the signal power [6]. The relation between the power $P(n)$ and the received signal $x(n)$ is shown in (3).

$$P(n) \;=\; \alpha P(n-1) \;+\; (1-\alpha)x^2(n) \tag{3}$$

where $0 < \alpha < 1$ is the location of the pole of the IIR filter. When α is close to 1, the current estimate of power depends more on the previous estimate of the power than on the instantaneous value of signal power and is more accurate. Thus, if α is pushed close to the unit circle, then the IIR filter gives an accurate estimate of the signal power.

3.2 Decision Logic Back-end

The output of the signal processing front-end is a binary "0" or binary "1". After obtaining the binary bits we need to decode them so as to obtain the correct time. This would include adjusting the day and time depending on the daylight savings time bit and leap year bit. The displayed time would also be changed depending on which time zone (eastern, central, pacific, mountain) was selected. This decision making logic is modeled in the Finite State Machine (FSM) domain.

4 Implementation on a Microcontroller

The radio clock modeled above was implemented on a PIC microcontroller from Microchip Technology Inc. We chose PIC16C7X family because of their low cost (about $6 in bulk) and its Harvard architecture (similar to the architecture of a DSP) [7]. This family of also has an on-chip analog to digital converter, thus reducing peripheral interface latency.

4.1 Decoding the time standard

The decoder for time standard is implemented using PIC16C71 which is a 18-pin microcontroller with an internal 8-bit analog to digital converter. All of the instructions are single cycle instructions (400 ns with a 10 MHz clock input) except for program branches which take two cycles [7]. It also has an 8-bit timer-counter and a watchdog timer. PIC16C71 has a high performance RISC CPU and has only 35 instructions.

The sampling frequency was chosen to be 6.25 kHz. 6.25 kHz is equivalent to a sampling interval of 160 μs, thus 400 instructions could be executed between two samples. A value

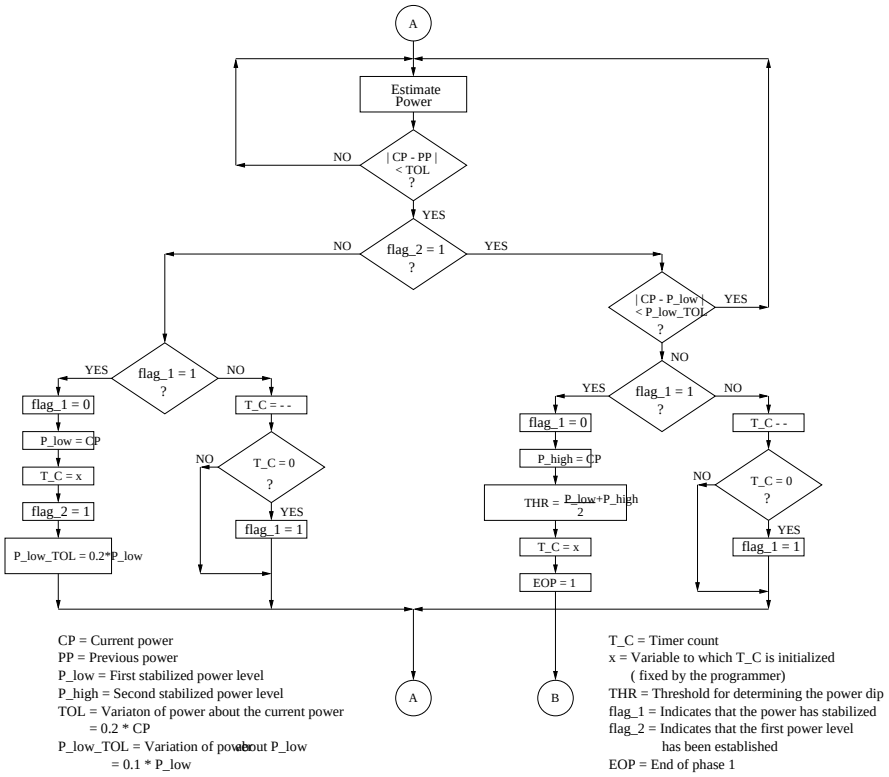

Figure 2: Flowchart to estimate the two power levels

corresponding to 400 was stored into the timer so that it would interrupt after 400 instruction cycles and sample the analog data.

The algorithm used for decoding can be broadly divided into three phases:

1. Estimate the two power levels P_{low} and P_{high} and set the threshold to be between them (Figure 2).
2. Track the start of the minute. Two consecutive *reference markers* indicate the start of a new minute.
3. Track the bits, decode the bits with appropriate weights (BCD), and calculate the local time.

Our technique has very low computational complexity. It requires only three multiplies per sample for phase 1 and two multiplies per sample for phases 2 and 3, respectively. All multiplications are between a variable and a constant and are optimized for length and run time. Table 1 summarizes the memory and computational power required.

	Program Memory	**Data Memory**	**Multiplies per sample**	**Divide per sample**
Decoding the time standard	800 words	22 bytes	3	0
Generating a frequency reference	500 words	15 bytes	1	1
Combined decoder	1300 words	40 bytes	4	1

Table 1: Worst case analysis of memory requirements and computational power.

4.2 Frequency Reference

Generation of a frequency reference requires a pulse-width-modulation (PWM) unit, and hence we choose PIC16C72 for this application. PIC16C72 has 2000 words (2000×14 bits) of internal memory and 128 bytes of data memory. It has a PWM unit and a 5-channel 8-bit A/D converter.

The PWM unit generates a square wave of specified period and duty cycle. The period and duty cycle counts are stored in two PWM registers. PWM unit has a prescaler and a postscaler. Depending on the prescaler value n, the timer is incremented every n cycles, until it matches the PWM period count. The timer starts counting from 0 and the PWM outputs high. When the timer value equals the duty cycle count, the PWM output is made low. When the timer value equals the period count, the PWM output is again made high and the timer is reset to zero.

Once the Friedman interpolator locks onto the signal and the frequency estimate stabilizes, the PWM unit is started at the first sample after a positive going zero crossing. At each positive going zero crossing, $\delta(n)T_s$ in Figure 1 is calculated. Since we know the period of the sinusoid accurately, we can calculate the time that this sample occurs from the first zero crossing time. This time can then be converted into an appropriate timer count by dividing it by the prescaler value n times the instruction cycle speed. This timer value is compared with the actual timer 2 value. Any difference between the two is stored. The difference may not be accurate per cycle because of the phase jitter. To remove the effects of phase shift, we keep a moving average of the difference between the calculated timer value and the actual timer value. At the end of one day, the difference is incorporated into the timer count. The whole process is then repeated.

5 Conclusions

Market forecasts [8] suggest that radio-controlled time-keeping will soon be as popular as the ubiquitous quartz watches and clocks. This project is aimed at providing a low-cost solution for an useful product that is used daily by many people. We present a new approach to decode the WWVB broadcast time information and to generate an accurate frequency reference calibrated to the NIST primary standard. This reference frequency has a relative frequency of at least 10^{-4} to 10^{-6} and can be used as a secondary standard. The key innovations are new, zero-buffering algorithms and microcontroller implementations to decode the WWVB time code information and to use a pulse-width modulator to generate an accurate frequency reference from the WWVB signal. We implement the time information decoder and the frequency reference generator on PIC microcontrollers. The decoders were tested on separate microcontrollers of the same family (PIC16C7X). A combined decoder can be implemented on a PIC16C72. The microcontroller based decoder is at least half the cost of the existing WWVB decoders. Our combined decoder for decoding the broadcast time information and generating an accurate frequency reference is the first to be implemented on a microcontroller.

References

[1] G. Kamas and M. A. Lombardi, *NIST Time and Frequency Users Manual.* Boulder, CO: National Institute of Science and Technology, 1990.

[2] T. E. Parker and J. Levine, "Impact of New High Stability Frequency Standards on the Performance of the NIST AT1 Time Scale," *IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 44, pp. 1239 – 1244, Nov 1997.

[3] NIST Boulder (Colorado) Laboratories, "http://www.boulder.nist.gov."

[4] V. Friedman, "A zero crossing algorithm for the estimation of the frequency of a single sinusoid in white noise," *IEEE Trans. on Signal Processing*, vol. 42, pp. 1565 – 1569, June 1994.

[5] P. E. Wellstead, "Aliasing in system identification," *Int. Journal of Control*, vol. 22, pp. 363 – 375, Sep. 1975.

[6] Z. Dusan, "Mean power estimation with a recursive filter," *IEEE Trans. on Aero. Electronic Sys.*, vol. 13, pp. 281 – 289, May 1977.

[7] Microchip Technology Inc., "http://www.microchip.com."

[8] BRG Atomic Time Clocks and Watches, "http://www.fiest.com/ bodegroup/arc1.html."