A Voice Time Monitoring and Recording Sub-System for the Telephone Speaking Clock

C.C. Lin*, P. C. Chang, J. L. Wang, T. Y. Chiu, and S. Y. Lin

National Time & Frequency Standard Lab., TL,
Chunghwa Telecom. Co., Ltd.
No. 99, Dianyan Rd., Yangmei City, Taoyuan County 326, Taiwan
*cchlin@cht.com.tw

Abstract—Previous research has developed a time source selecting and data monitoring system (TSMS) for the time synchronized speaking clock (TSSC) [1]. This developed includes two sub-systems. One is for multi-time signal source selecting and the other is for input status monitoring.

In this research, the TSMS was extended to monitor the output of the TSSC on dedicated telephone line, called TSMS_A1. From this sub-system, an interrupt circuit with a Linux kernel driver was developed for capturing the rising edge of the on-time tone which is from TSSC output to produce an interrupt event. Through this event, the TSMS' controller was trigged immediately to measure the time error and record the announced voice program into an audio file. The audio recording procedures and a raw file converting from PCM (Pulse Code Modulation) into Waveform Audio File Format (WAVE) were studied. In the system design, Unified Modeling Language (UML) notation tools, such as user case diagram and activity diagram were used to express basic concepts of the system requirements and interactive behaviors.

Finally, the data were analyzed from the on-time tone measured by setting three typical threshold levels. Comparing these results, optimum one was chosen as the input threshold for the TSMS_A1. Its time comparison error is within ±100μs after the path delay has been removed. From this result, we could indeed monitor the output status of the TSSC and then could be providing more reliable voice time service for users.

Key words: speaking clock, monitoring system, recording system, interrupt, UML, PCM, RIFF, WAVE, OTM

I. INTRODUCTION

In Taiwan, the speaking clock can be reached by dialing '117' on a telephone line. A recorded female voice says (for example): "下面音響 10 點 20 分 30 秒" followed by a tone (beep) of 800Hz, 250 ms long, as shown in Figure 1. The meaning is, at the rise edge of the beep, the time will be 10 hours, 20 minutes and 30 seconds. The time is announced in 10 second intervals. The TSSC's broadcasting center is located in Taipei City, which is north of TL (Chunghwa Telecom. labs in Taiwan) by about 50 kilometers. In the center, the TSSC is synchronized with the national standard time by a telephone dedicated line, on 24 hours a day and 7 days a week.

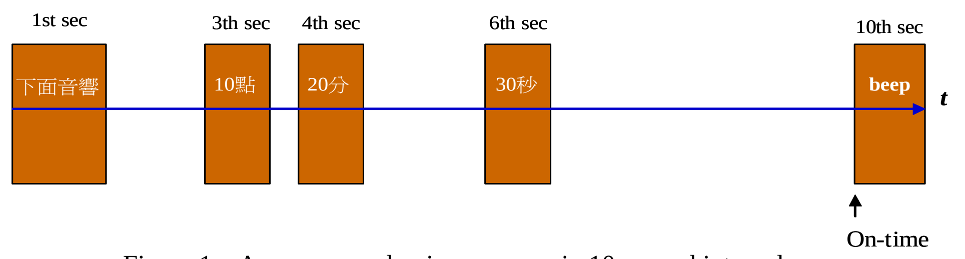


Figure 1. An announced voice program in 10 second intervals.

In the previous research, the TSMS, only aimed at the input signal of the TSSC, was developed as shown in Figure 2, labeled "A" and "B." Two major tasks are as follows. First, set up a multi-time selecting system which can switch to a backup time signal source automatically when the present one detects an error. Second, establish data monitoring which helps the operator check the condition of input synchronized time signals. When the system is detected in abnormal condition, it may trigger an alarm immediately. However, the TSMS does not have the output status monitoring functions for the TSSC yet. Thus, the objective of this research is to extend the functions of the TSMS to monitor the TSSC output signal labeled "C." It includes recording the announced voice program and measuring the time error of the on-time tone.

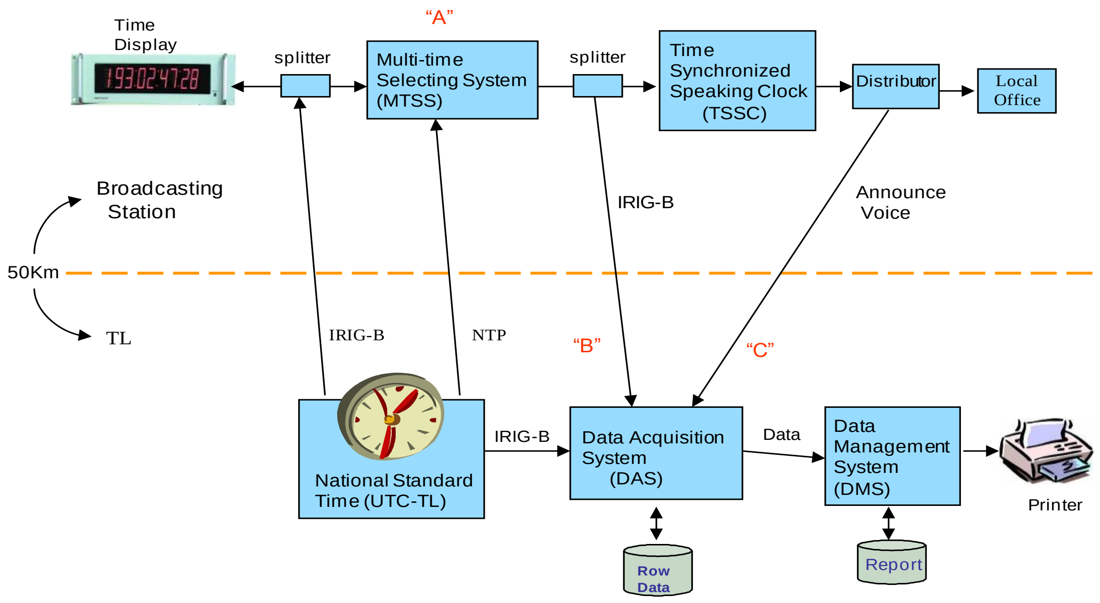

"A"
Time Display
splitter
Multi-time Selecting System (MTSS)
splitter
Time Synchronized Speaking Clock (TSSC)
Distributor
Local Office
Broadcasting Station
IRIG-B
Announce Voice
50Km
TL
IRIG-B
NTP
"B"
"C"
National Standard Time (UTC-TL)
IRIG-B
Data Acquisition System (DAS)
Data
Data Management System (DMS)
Printer
Row Data
Report

Figure 2. Whole block diagrams of the speaking clock monitoring system.

II. REQUIREMENTS ANALYSIS

The basic idea of the TSMS_A1 could look like Figure 3. First, the input signal is separated into two paths flowing through the measuring part and the recording part. In the measuring part an on-time pulse should be generated by a filter and a shaper comparing with a standard time. In the recording part it should amplify the input signal and then an announced voice program is recorded by a recorder. At last, all data need to be saved into a store device for analysis after operating. The main functions of the TSMS_A1 are going to be considered in the flow diagrams:

- Every 10th second, capture the on-time tone from the announced voice program through the telephone line.
- Measure the time error of the on-time tone comparing to the standard time and then save the data to an external storage device.
- Every 10th second, record the announced voice programs into an audio file, and then save the file to an external storage device.
- The measuring and recording mechanics should be stable and reliable.
- Be of good sound quality, and easy to replay.
- A controller should work under the Linux operating system to meet the previous TSMS' operating environment.

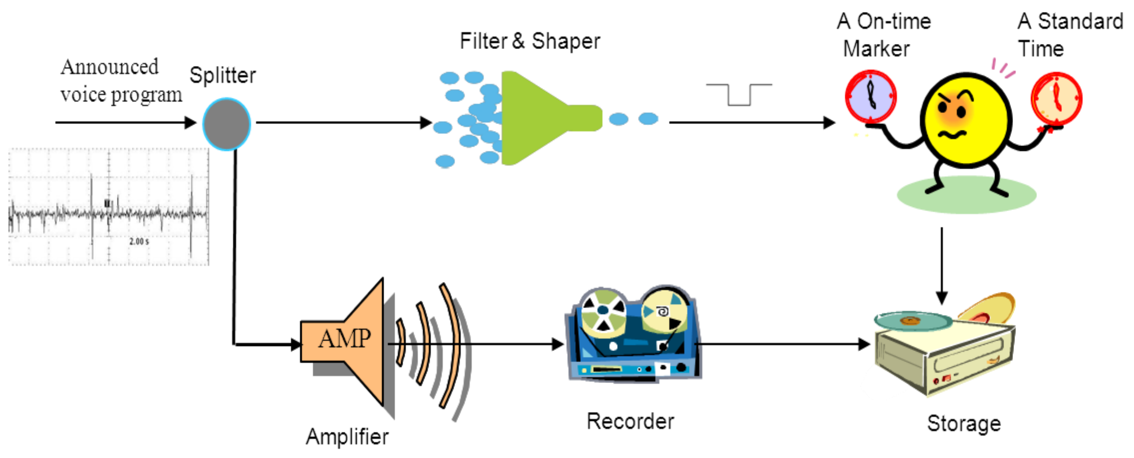


Figure 3. Abstract statement diagrams of the requirements of the TSMS_A1.

A. *User Cases*

Use cases describe the functionality of the system from the user's point of view [2]. In the TSMS_A1 there is an event and then many different use cases, as shown in Figure 4.

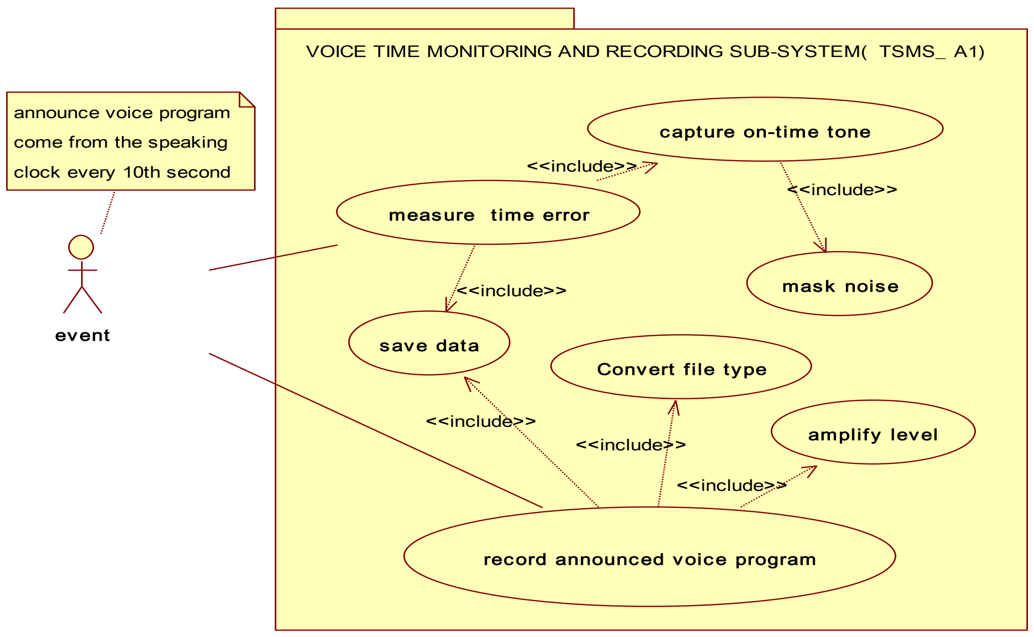


Figure 4. The user case of the system requirements of the TSMS_A1.

1) Event: In this first stage of the analysis, consider the system is a black box reacting to the requests and message from the environment. An event is an important message from the environment. A real-time reactive system has to react to the external events in a bounded time.

2) Measure time error: Describe how to start a job that measures the time error comparing the on-time tone with the standard time.

3) Capture on-time tone: Describe how to capture the on-time tone from the announced voice program.

4) Mask noise: Describe how to filter the noise from the the input signal to keep an on-time tone only.

5) Record announced voice: Describe how to provoke a job that records all announced voice programs.

6) Amplify level: Describe how to amplify the weak voice for the recorder.

7) Convert file type: Describe how to convert the audio file from raw PCM to WAVE.

8) Save data: Describe how to save the data to the external hard disk including time error and announced voice programs.

B. Scenarios

This describes the interaction between the external event and the TSMS_A1 system. Here, two main parts are described. Figure 5 shows the scenarios for the Measure Time Error user case. In 10 second intervals, the event triggers the Noise Masker to mask the announced voice program from the 1st to 9th second except for the 10th second, an on-time tone. The only on-time tone flows to the Interrupt Pulse Generator producing a sharp pulse which starts the interrupt action of the controller to execute a service subroutine. In this time, the subroutine plays the role of calculating the time error and then saves these data to an external hard disk. Then, the system waits for the next event spaced 10 seconds again.

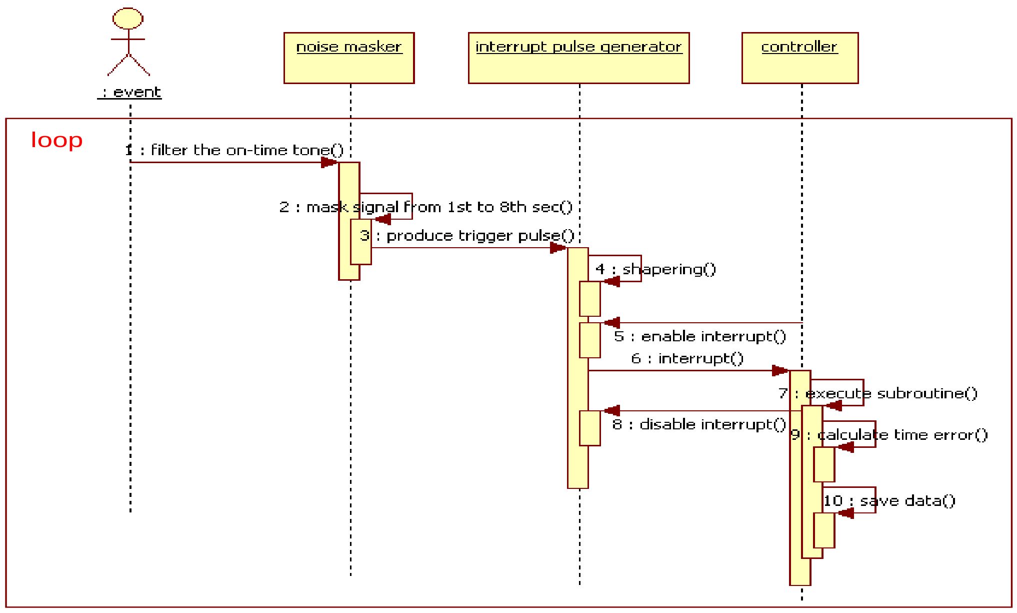


Figure 5. The interactive behaviors of the Measure Time Error user case.

Figure 6 shows the scenarios for the Record Announced Voice Program use case. Interactions between the external event and the record system are described. First, the Amplifier magnifies the event level to feed the soundcard which is plugged into the controller's slot. Then, the Controller converts the audio file type from PCM into WAVE format. After the file type converted completely, it should save the audio file into an external hard disk. Then, the system waits for the next event again.

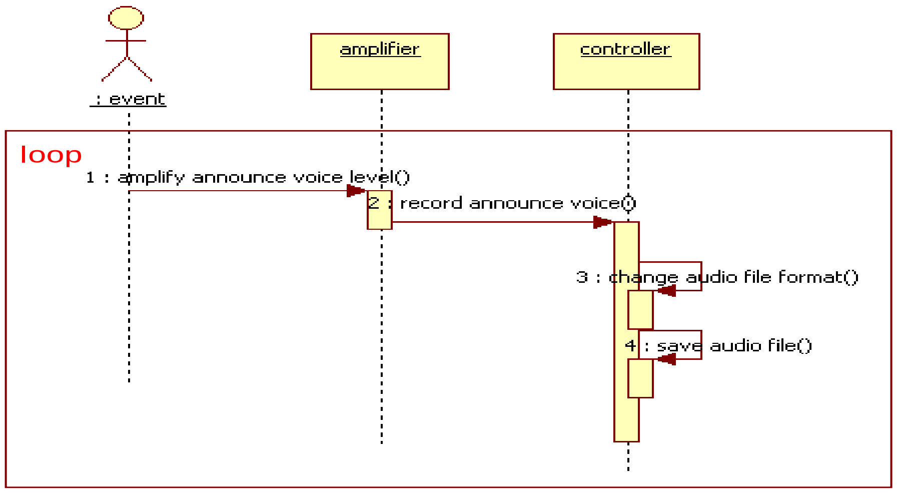

: event
amplifier
controller
loop
1 : amplify announce voice level()
2 : record announce voice()
3 : change audio file format()
4 : save audio file()

Figure 6. The interactive behaviors of the Record Announced Voice Program user case.

III. Hardware Design

In Figure 7, the whole hardware structure is shown including a capture and amplifier circuit added to the TSMS' controller. First, the input announced voice striking an audio transformer (ST-71) induces two current paths. One flows into the on-time tone measuring leg and the other flows into the announced voice recording leg. In the measuring leg, an operational amplifier (OP) is used to construct a SCHMITT trigger circuit, which shapes the input announced voice to produce a bipolar pulse in 10 second intervals when the input signal level reaches the threshold in advance setting. At this OP (1/2 TL082) output, the bipolar pulse is rectified into a unipolar pulse by a diode (IN4148). Then, the pulse flows into an interrupt port (labeled INT5) of the controller. The controller starts to execute a service subroutine for calculating time error of the on-time tone comparing with the standard time. In the recording leg, the input voice level is increased by the other OP of the TL082 IC. Then, the magnified voice flows into the audio input port CD-IN which is one of the sound card audio inputs. At the same time, the controller executes an application program to record and store the audio file onto a hard disk every 10 seconds.

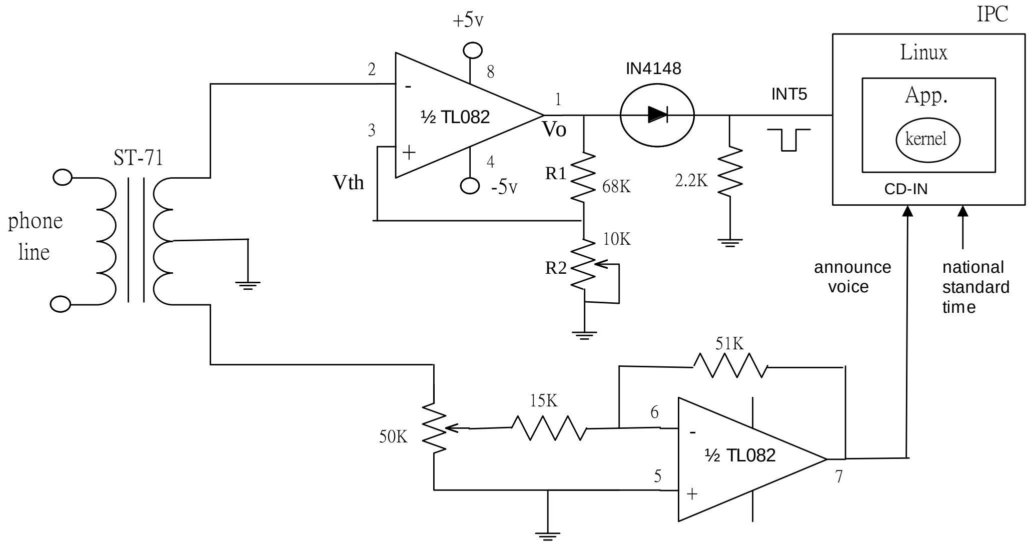


Figure 7. The hardware structure and the circuit of the TSMS_A1.

IV. SOFTWARE DESIGN

In the scheme of software design of the TSMS_A1, two software programs including a kernel driver and an application program are designed under Linux operating system in C language. It is shown by three thick line blocks in Figure 8. An application program is mainly concerned with processing including the time error calculating, the audio file recording and its data format converting from the raw PCM to WAVE. In kernel space a driver, including Get/Set time functions for an IRIG-B time card and Enable/Disable interrupt functions for the controller, was designed. Generally, I/O hardware devices can only be accessed through kernel programs called device drivers. Direct access to the devices is not permitted by the operating system, so an application program has to go through device drivers to gain access [3][4].

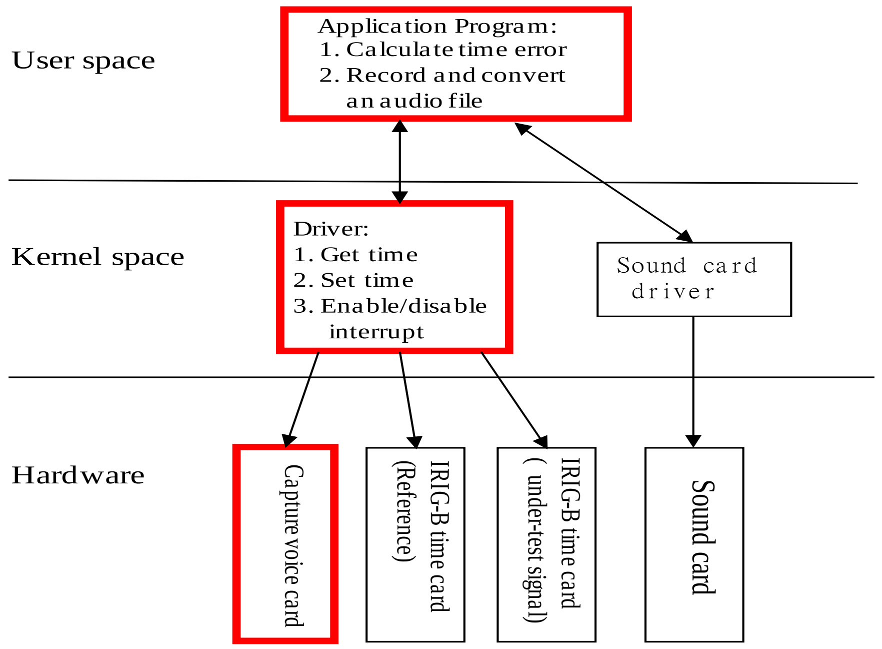


Figure 8. Three main design blocks of the TSMS_A1 on the Linux operating system.

A. *Kernel driver*

Figure 9 shows the state chart consisting of 7 states to model the behavior of the kernel driver in response to the application program (APP) or the external event. It includes a time process function and an interrupt service routine to measure the time error of the on-time tone. When the power turns on, it causes a transition to the Initial state. Subsequent is the ready event, which transitions from the Initial state to the Waiting for APP control code state. During this state, the driver is waiting for APP to input the control code. When the control code enter event occurs, the driver transitions to the Validating control code state. In this state, the driver determines whether the control code matches the stored control code. If the control code does not match, the invalid control code is taken back to the Waiting for APP control code state. If the control codes match, there are two possible state transitions out of the Validating control code state. One is the valid APP timestamp at the 9th second control code which transitions to the Enable interrupt state. From this state, the external on-time pulse event causes a transition to the Disable interrupt state. The measure event is depicted after the Disable interrupt state, which causes a transition to the Get reference time data state. The other event is the valid APP timestamp at 1st second control code, which will transition to the Write data to APP state. At the next event, it will transition back to the Waiting for APP control code state.

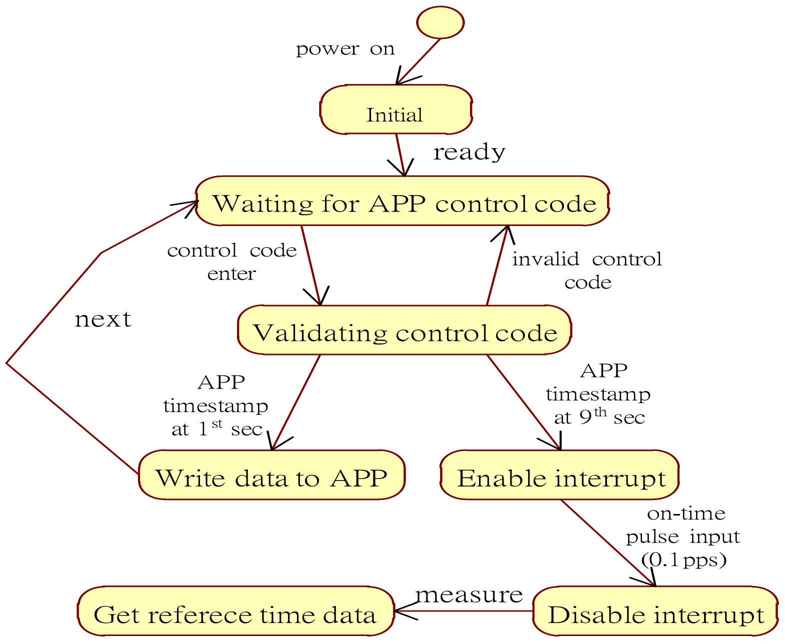


Figure 9. The state chart of the kernel driver.

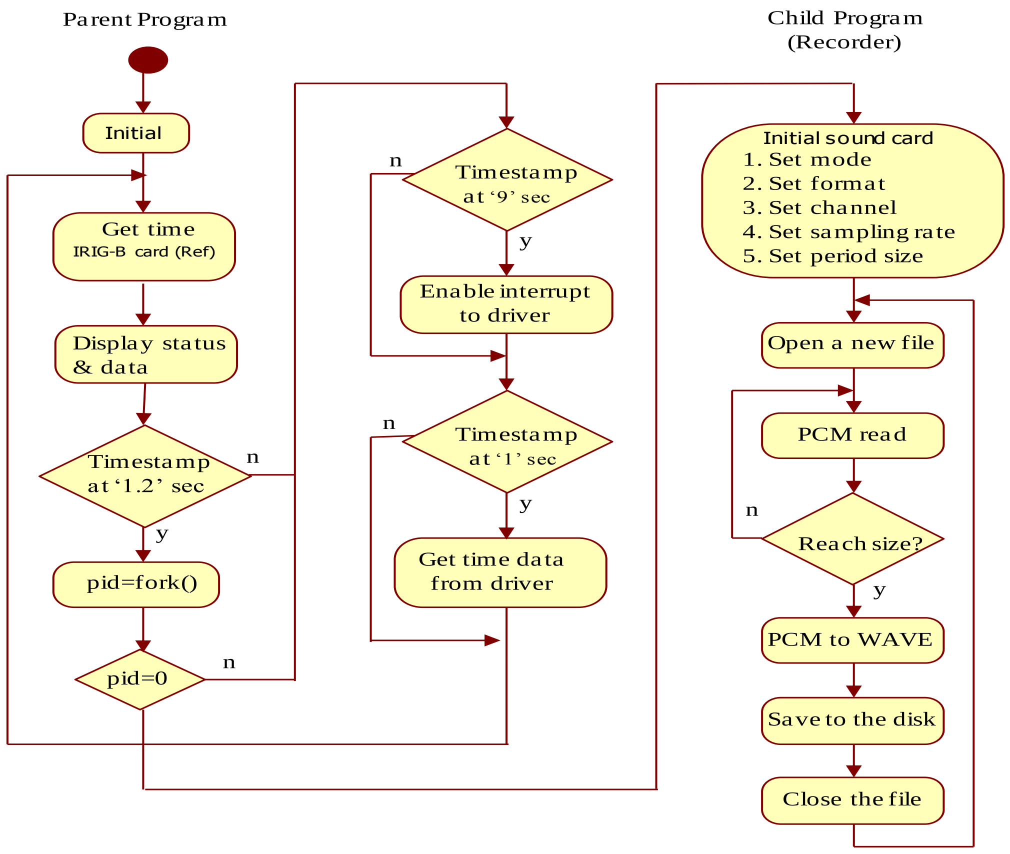


Figure 10. The flow chart of the application program.

B. *Application programs*

Figure 10 shows the flowchart for the application program to monitor the output of the TSSC. This flowchart with a loop is started by entering a parent process which performs a role of a status monitor, and then creates (forks) a new child process every 10th second, which performs the role of recording the announced voice programs. In the parent process, the system current operating status and the measured data is displayed in the screen. The decision symbol tests whether the timestamp is at the 9th second. If the answer is "Yes," it enables an interrupt operation in the kernel driver. Next decision symbol tests whether the timestamp is at 1st second. If the answer is "Yes," it starts to get time data from the kernel driver and then stores these data to a disk. In the child process, a recording program first initializes a sound card including mode, format, channel, sampling rate, and period size to perform a role of an audio recorder. Then, it opens a file and reads audio raw data in PCM format. Next, the decision symbol tests whether the data size has been reached in full. If the answer is "Yes," it converts the audio file from PCM into WAVE and then stores the file to a disk. Finally, it closes the file and ends the child program.

C. *Audio file convertion*

To develop the announced voice recorder, an off-the-shelf sound card with kernel driver plugging into the controller's slot is used and a linked Advanced Linux Sound Architecture (ALSA) library implements required functions. ALSA consists of an application programming interface (API) library, a kernel driver and utility programs for sound operations under the Linux operating system. PCM is a digital audio processing technique with samples generated in continuous time periods. To write a PCM application for ALSA, first it needs a handler for the PCM device and then some information about the configuration like buffer size, sample rate and the PCM data format. In this development system, it first stores the audio file to an external storage device and then converts its raw PCM file to a WAVE file. A WAVE format is a subset of Resource Interchange File Format (RIFF), which can include many different kinds of data. It was intended for multimedia files originally, but the contents are open enough to enable more things to be placed. A RIFF file starts out with a file header followed by a sequence of data chunks. The structure of a RIFF file looks like follows [5]:

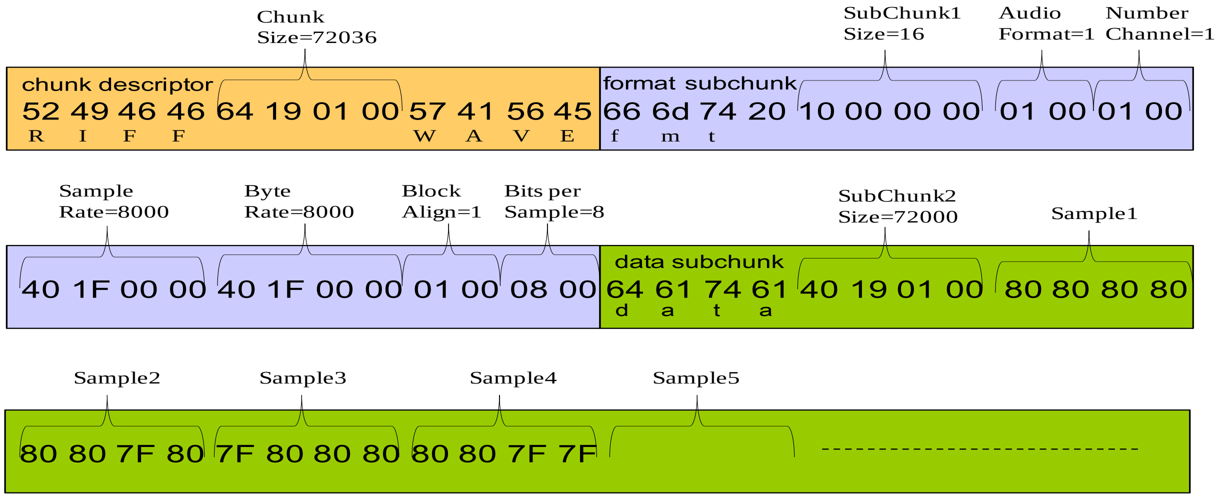
Offset	Description
00	'R', 'I', 'F', 'F'
04	Length of the entire file (32-bit unsigned integer)
08	form type (4 characters)
12	first chunk type (4 characters)
16	first chunk length (32-bit unsigned integer)
20	first chunk's data

The WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks: the format chunk which describes the sample rate, sample width, etc. and the data chunk which contains the actual samples, i.e. the raw data of the PCM. Contents are stored in the low-high byte ordering (little-endian) or in the high-low byte ordering (big-endian). Table 1 [6] illustrates the Canonical WAVE file format. As an example, one of the recorded WAVE files is described in decimal numbers as shown in Table 2 [6]. The size of a record is about 7Kbyes.

Table 1. The Canonical WAVE file format.

Field offset (bytes)	Field name	Endian	Field size (bytes)	Description
0	ChunkID	big	4	Contains the letters "RIFF"
4	ChunkSize	little	4	36+SubChunk2Size
8	Format	big	4	Contains the letters "WAVE"
12	SubChunk1ID	big	4	Contains the letters "fmt"
16	SubChunk1Size	little	4	16 for PCM
20	AudioFormat	little	2	PCM=1 (Linear quantization)
22	NumberChannels	little	2	Mono=1, Stereo=2, etc.
24	SampleRate	little	4	8000, 44100, etc.
28	ByteRate	little	4	SampleRate x NumChannels xBitsPerSample/8
32	BlockAlign	little	2	NumChannels xBitsPerSample/8
34	Bits/Sample	little	2	8 bits =8, 16bits=16, etc.
36	SubChunk2ID	big	4	Contains the letters "data"
40	SubChunk2Size	little	4	NumSamples x NumChannels xBitsPerSample/8
44	Data	little	SubChunk Size	The actual sound data

Table 2. A recorded WAVE file of the TSMS_A1 real operation.



V. Experimental Results

To get optimum performance for the TSMS_A1, choosing a suitable threshold voltage in the capture circuit is very important because it would affect measuring results. First of all, it is necessary to examine the vibration characteristics of the original interrupt-processing of the controller, bypassing the capture circuit and directly connecting 1PPS (a standard time) into the interrupt port of the controller. Then, executing an interrupt service routine calculates the time error. Figure 11 shows the histogram of estimated. Its standard deviation is equal to 1.3μs and the mean approaches zero. Therefore, the effects of the latency may be neglected in terms of the sub millisecond grade in following tests.

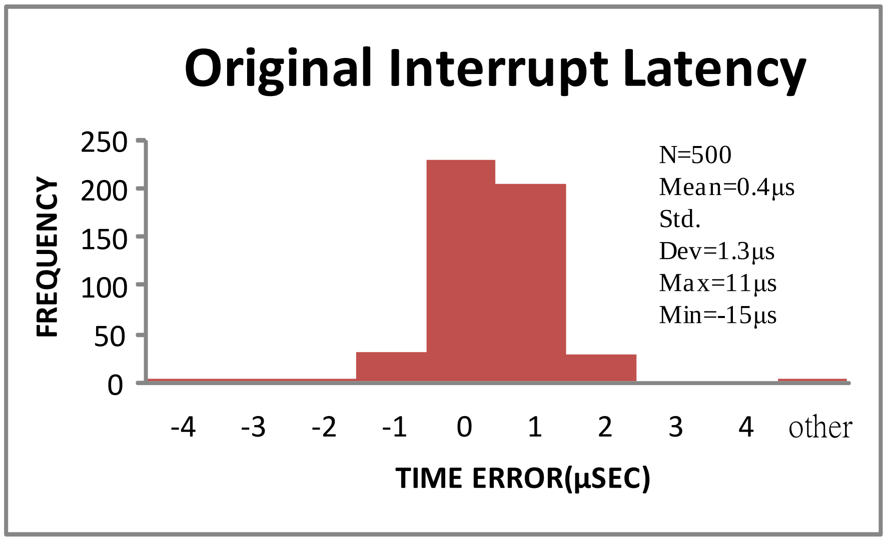


Figure 11. The time error of an original interrupt test.

For the capture circuit in Figure 7, the input signal level in voltage is between -0.6V and +0.6V from the dedicated phone line. Three representative examples of the threshold level are described respectively in Figure 12. It was calculated according to following formula:

$$V_{th} = V_o \times \frac{R_2}{R_1 + R_2}$$

Where V_{th}: threshold voltage

V_o: output voltage

1) Low level:

V_{th} is set at 0V, adjusting R_2 to zero Ω. Its x-y plot is shown in Figure 12 labeled "Low." From this plot, some spikes are burst up by noise due to the lower threshold. The standard deviation is equal to 100μs and the mean approaches 1.4ms.

2) Middle level:

V_{th} is set at +0.16V adjusting R_2 to 220 Ω. Its x-y plot is shown in Figure 12 labeled "Middle." The standard deviation is equal to 48μs and the mean is equal to 2.6ms. There is not any spike in this line.

3) High level:

V_{th} is set at +0.34V, adjusting R_2 to 470 Ω. Its x-y plot is shown in Figure 12 labeled "High." From this plot, it has two boundaries, high within 3.75ms and low within 2.50ms. It means that sometime, the first cycle of the input signal level cannot reach the threshold, but the second cycle may reach it. This ambiguous interval is just in 1.25ms (800Hz) shown in Figure 12.

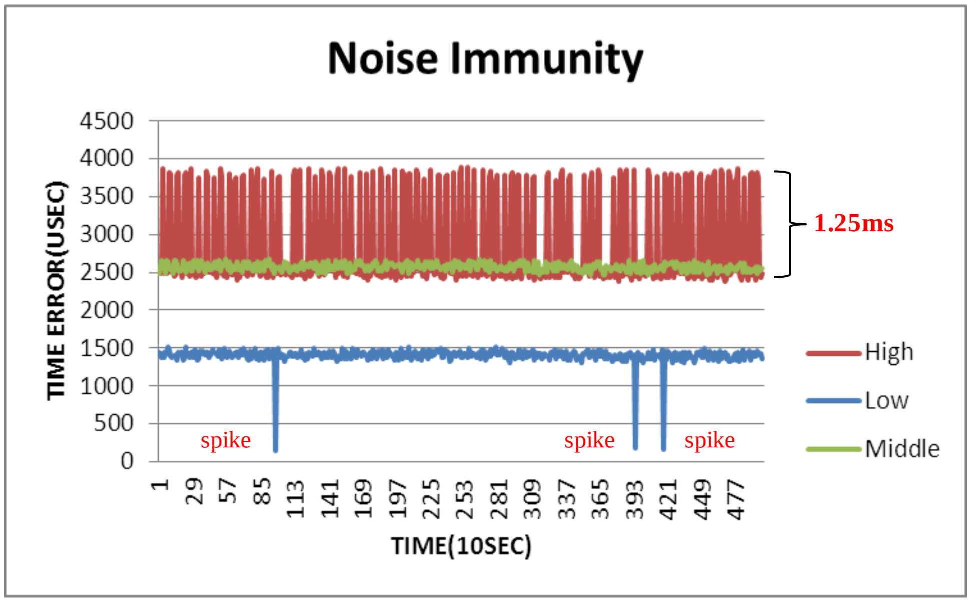


Figure 12. Noise immunity using three difference threshold levels.

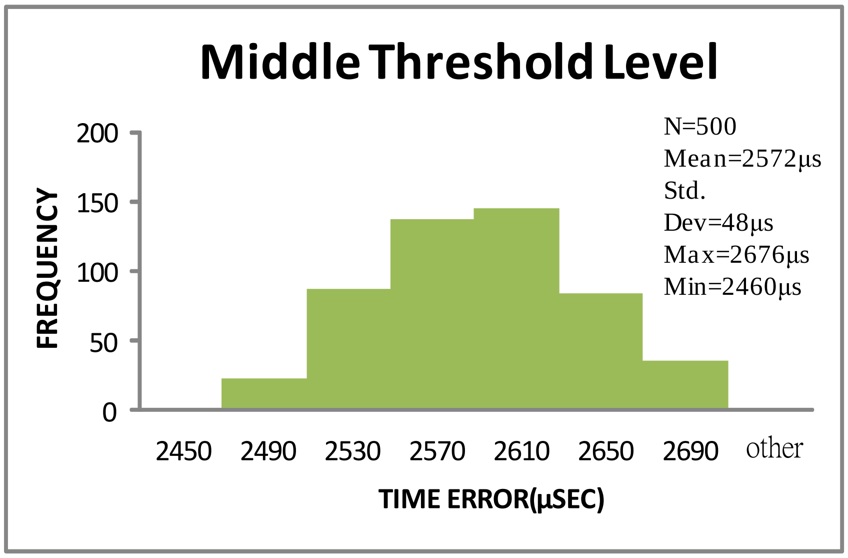


Figure 13. Histogram of the optimum one threshold chosen.

VI. CONCLUSION

In this research, a sub-system was developed aimed at the output monitoring of the speaking clock. Using an on-time tone capture circuit and a PC-based interrupt faculty operated under controlled conditions, the time error was effectively measured. Three curves in Figure 12 show obvious difference of the mean because of the vary threshold chosen. Also spikes, ambiguous points are appearance in the low and high threshold settled. These results indicate that the optimum threshold for the TSMS_A1 design is the "middle" level to achieve the maximum efficiency. The histogram of the middle threshold is shown in Figure 13. This present a path delay of 2.57ms, a standard deviation is less than 50μs. The time comparison error is within ±100μs after the path delay (mean) has been removed. It is allowing a sufficiently accuracy for the speaking clock monitoring. At the same time, whole announced voice program was recorded and stored into an external storage in WAVE format. It can be traceable in after days when an error broken. From this research, we could indeed help the operator of the speaking clock service to make informed device-replacing decisions. We also recommend that these experiments could be useful in measuring the accuracy of time facilities in the sub-millisecond range.

REFERENCES

[1] C. C. Lin, P. C. Chang, J. L. Wang, and S. Y. Lin, "Design and Implementation of a Time Source Selecting and Monitoring System for the Telephone Speaking Clock," Proc. of 41st PTTI, 2009.

[2] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, and K. A. Houston, "Object-Oriented Analysis and Design with Applications," Third Edition, Addison Wesley Professional, 2007.

[3] J. Corbet, A. Bubini, and G. Kroah-Hartman, "Linux Device Drivers," 3rd Edition, O'Reilly Media, Feb. 2005.

[4] Yutaka Hirata, "Linux Device Driver Programming," Traditional Chinese edition, DrMaster Press Co., Ltd. Mar. 2009.

[5] *http://www.lightlink.com/tjweber/StripWav/WAVE.html*

[6] *https://ccrma.stanford.edu/courses/422/projects/WaveFormat/*