Chapter 4

Speech Synthesizers

An essential step in the study of pathological voices is re-synthesis; clear and immediate evidence of the success and accuracy of modeling efforts is provided by comparing the original and synthetic versions of the pathological voice. The effects of variations of each of the model parameters may be quickly evaluated perceptually by generating synthetic voice samples with an easily controlled synthesizer. Tests may be performed to validate analysis results, and experiments may be performed to determine the effects on the listener of variations and interactions of model parameters. This chapter describes the implementation of two synthesizers used in the study of pathological vowels: a real-time hardware synthesizer, and a software synthesizer implemented in MATLAB [26]. In contrast to most voice synthesizers, the synthesizers implemented in this study were specialized and optimized for the generation of pathological vowels for the purposes outlined in Chapter 1. Ordinary speech synthesis for commercial purposes generates connected speech of (hopefully) reasonable quality; here, the synthesizers were designed with the model parameters described in vowel analysis (Chapter 2), and they evolved to high levels of fidelity. In some cases, the synthetic and original vowels are indistinguishable.

4.1 Hardware Real-time Synthesizer

The first efforts at synthesis were directed at construction of a real-time vowel synthesizer based on the source-filter (Fig. 1.2) model of voice production. The primary motivation was to provide a system capable of immediate feedback when changes were made to model parameters, such as formant frequencies and bandwidths, fundamental frequency, and noise levels. In order to quickly implement the system, readily available components were employed; a PC (personal computer) platform with an X86 (Intel 80X86 processors) CPU was selected. Hardware subsystems were implemented on a generic ISA (Industry Standard Architecture) bus [33] prototyping adapter PCB (printed circuit board) using commonly available IC's (integrated circuits). Software for real-time performance was written in high-speed assembly language and was designed to disable the MSDOS (Microsoft Disk Operating System). In this section, the design philosophy of the real-time synthesizer is presented, along with an overview of its hardware and software implementation.

4.1.1 Real-time and Control Concepts

In order to provide immediate and accurate synthesis with real-time response, two requirements must be met:

1. The results (synthesized sound) must be produced quickly. The results of all calculations must be performed within a time budget set by the sample period selected (usually 0.1 ms.). Updated output waveform sample points are computed based on time varying model parameters such as fundamental frequency, noise levels, etc., and the new values must be determined within the time available (one sample period of 0.1ms or one “frame time” of about 20 ms). This speed requirement was achieved with X86 processors running assembly language code.

2. The results must be supplied deterministically. That is, successive executions of the calculations must occur on a repeatable schedule, and there must be no variation in the regularity (“jitter”) in the production of sample outputs to the output device (speaker or headset). Precise periodic output was achieved by keeping the computation periods within the cycle or frame budget and using crystal-controlled hardware clocking to latch outputs from the D/A (digital to analog converter). Non-deterministic behavior due to operating system activity was achieved by deactivating MSDOS, which launches interfering tasks.

Another design principle incorporated into the real-time synthesizer was the capability to perform all computations within a single sample period; the CPU is phase-locked to the sample period. Most voice synthesizers using PCs either do not produce real time output at all (e.g., Klatt [32]) or allow the CPU to run unsynchronized, filling buffers for later clocked output. Such asynchronous computation is acceptable only if synthesizer output is to be generated, as resulting latency delays will be below perception limits in well-designed systems. However, if real-time control with dynamic inputs and feedbacks are to be incorporated, the CPU must process all feedback signals and new sensor inputs and generate updated control outputs within a single sample period, as illustrated in Fig.4.1 and Fig.4.2. Buffering and variable delays would render control algorithms unpredictable. By keeping all computations within a single sample period, the system is rendered expansible to a variety of possible closed loop controls applications. For example, slidepot controllers could be used as input devices to control a bank of formant frequencies and bandwidths, allowing very rapid evaluation of vowel quality variations.

Single cycle computation capability was maintained throughout the development of the real-time synthesizer, despite the ever-increasing computational demands of the addition of new features to the synthesizer. The evolution of the X86 processor line from 286 to 386 to 486 to Pentium kept throughput up to the increasing computational tasks, as shown in Fig. 4.3.

4.1.2 Functional Overview

An overview of the functions of the alpha implementation of the hardware synthesizer is displayed in Fig. 4.4. This version was a simple proof of concept test bed, using commercially available adapter cards and external prototyping electronics rather than the custom wirewrap adapter card of the final version. The alpha implemented up to five formant resonators with a simple impulsive source and generated acceptable vowel sounds. An 80286 platform was used as the processor. Timing was provided by a Metrabyte CTM05 adapter card using an AM9513 counter-timer to provide a 10 kHz clock output for ISR (interrupt service routine) interrupts to the processor to initiate each control cycle and to strobe the calculated output into the D/A (digital to analog converter) data latch. A reconstruction filter (low pass filter to remove "steps" from the D/A output), audio amplifier, and loudspeaker completed the issue. The impulsive source and resonators were implemented in X86 assembly language, using 16 bit fixed point arithmetic for all digital signal processing. The 16 bit fixed point arithmetic resonator calculations had to be painstakingly scaled to prevent integer over/underflows.

Many upgrades and refinements resulted in the real-time synthesizer in its current form, as shown in Fig. 4.5. The impulsive source was generalized to include the KGLOT88 [22] and the LF [12,31] models for the glottal source flow derivative waveform. These resulted in an immediate improvement in the realism of the synthetic vowels. In addition, provision was made for inclusion of an arbitrary wave for the source, allowing offline generation of any desired waveform. Aspiration noise simulation was added to the model, again improving realism. Numerator zeros were added to the vocal tract model to allow simulation of nasal effects. An AGC (automatic gain control) function was implemented to compensate changes in output signal magnitude due to parameter changes, thus preventing overflow of the D/A but retaining full use of its input range. Provisions were added to allow time-scheduled variations in control parameters. A flexible scheme for saving and recalling control parameters and signal time histories was implemented. Provisions were made to generate nonperiodic variations in frequency (jitter) and amplitude (shimmer), and to generate diplophonia (alternate cycle variations in period and amplitude). All hardware functions were consolidated on an ISA prototyping adapter card implemented with wirewrap. The synthesizer was initialized under MSDOS 6.2 on a Pentium 120 MHz or higher platform. In the following sections, the hardware and software implementations of the real-time synthesizer are briefly described.

4.1.3 Hardware Implementation

In order to achieve true real-time performance, it was necessary to provide hardware extensions to the PC platform. Efforts to use components native to the motherboard alone proved unsuccessful. The resulting hardware extensions resulted in a final issue of 17 integrated circuits on a wirewrap PCB adapter card plugging into the PC system bus. Interface of the synthesizer hardware to the PC is via the 16-bit ISA bus, as shown in Fig.4.6. The hardware components of the real-time synthesizer consists of the following subsystems:

1. An independent clock generator. In order to create precisely timed output samples from the D/A, it is necessary to supply an accurate timing signal to latch each new output from the synthesizer calculation. Software timing schemes are unreliable and are subject to jitter (variations in cycle to cycle period lengths). An easily usable source of timing

signals for this purpose was not available on the PC motherboard. Instead, an independent programmable, crystal-controlled generator was used. The generator is initialized with the divider ratios to produce the desired clock frequency (usually 10kHz).

2. D/A converter. This device converts binary 16-bit integer output from the synthesizer into an analog voltage to drive the reconstruction filter. A 2R ladder type converter was selected for its easily used high speed parallel interface to the data bus.

3. Timing and control. This subsystem provides the “glue logic” and control signals for synchronizing the CPU and with the various hardware subsystems; it was implemented in SSI (small-scale integration) TTL (transistor-transistor logic) and PAL (programmable array logic) devices.

Briefly, in operation the hardware subsystems perform as follows:

1. The clock generator is programmed with the proper control words and divider ratios to supply the basic synthesizer sample rate (usually 10kHz). This rate then determines the critical cycle time budget for one cycle control (usually 0.1ms).

2. The PC’s operating system is then disabled and the PC’s interrupt controller is reprogrammed to look at interrupts from the synthesizer.

3. The timing and control circuitry is then enabled via the control port to begin sending the clock pulses as interrupt signals to the CPU.

4. As each output sample is generated by the real-time software, it is written into the D/A latch when it becomes available (which must occur prior to the end of each control cycle).

5. The next clock pulse simultaneously initiates a new control cycle and enables the latched D/A value as an analog output.

6. The hardware continues to generate timed analog outputs until the system is stopped.

4.1.4 Software Implementation

The real-time synthesizer software performs generation of output time series simulating the voice signal. It is responsible for initializing the system, performing user interface, calculating each new voice sample output within the cycle budget (usually 0.1ms), and detecting error conditions such as cycle overrun (the software takes too long to calculate the next sample). A brief overview of the software is displayed in Fig. 4.7.

The core of the program is the ISR (interrupt service routine), which responds to each external clock pulse to begin each new control cycle (right side of Fig. 4.7). Each cycle performs all necessary calculations to generate the next output sample for the synthetic voice; these tasks include:

1. Update the current time counter.
2. Generation of current values of any time varying parameters, such as variable formant frequencies or user supplied schedules.
3. Generation of the current value of the source signal in the source-filter model, according to the current time and the interpolated value on the type of waveform being generated and any random noise component
4. Updating the 2nd order digital resonators that simulate the vocal tract.
5. Generation of the next voice signal output value.

6. Performance of the AGC (automatic gain control) algorithm.

7. Writing the final scaled output signal into the D/A converter input latch.

8. Detection of control cycle overrun. The user is warned when the CPU takes longer to calculate the next output sample than there is time available in the control cycle budget.

As discussed in Section 4.1.3, the ISR is invoked by a hardware interrupt to the CPU from the external real-time hardware clock, which is free from any jitter (period variation). The ISR must complete all necessary computations before the next interrupt. All ISR code was written, therefore, to attain maximum speed. Assembly language was used to directly control the X86 processor hardware to avoid inefficiencies imposed by a compiler. Despite often-heard comments to the contrary, C language proved to impose a fairly consistent speed penalty of about a factor of 10. In addition, high-speed coding techniques were adopted wherever possible. These included use of table lookups (rather than computations), replacement of loops with simple repetition of code blocks, use of pointers to minimize data movements, and the use of the numeric coprocessor for all arithmetic. By use of these techniques and the advances in throughput of the X86 processor line, it was possible to achieve a flexible vowel synthesizer with instant response to user inputs.

The remainder of the real-time synthesizer software, illustrated in the left side of Fig. 4.7, consists of initialization and background tasks. The initialization tasks prepare the PC for real-time operation and set up control parameters for the synthesizer. Prior to operation, it is necessary to disable the extant PC operating system (MSDOS) and interrupt control scheme, which is inherently useless for hard real-time applications due to its intrinsic non-deterministic (inconsistent) behavior. The operating system's interrupts are disabled and replaced with the hardware interrupt from the crystal-controlled clock by re-initializing the 8259 interrupt controller (or its ASIC functional equivalent in later PC's) and other operations. Similarly, conditions for normal operation are restored at the termination of the synthesizer execution, thus eliminating the need to reboot the PC.

In addition to initialization, the background software handles the GUI (graphical user interface) to process keyboard commands entered during operation of the synthesizer. Upon completion of the ISR, control returns to the background task, which performs lower priority functions, including:

1. Detection of keyboard inputs.
2. Writing new display values to the CRT.
3. Loading input data.
4. Writing output data.

The background software was written in C, to take advantage of its ease of use and more powerful programming features, since speed is a secondary requirement here.

4.2 Software Synthesizer

Following successful implementation of the real-time hardware synthesizer, effects concentrated on refinements of the synthesis algorithms with the goal of achievement of ever-higher levels of fidelity and match to the original voice (see Fig. 6.1). Concentrated efforts were applied to the source waveform and the nonperiodic features of the voice, including FM, AM, and aspiration noise. In order to more quickly evaluate algorithms for synthesis and to allow for the implementation of complicated procedures at the sacrifice of real-time response, a purely software version of the synthesizer was implemented in the MATLAB [26] interpreter. MATLAB permits matrix objects to be created and manipulated with high-level functions designed for controls and signal processing applications, as well as supplying easy to use graphical output. It has become an industry standard for signal processing analysis.

In the following section, an overview of the software synthesizer is presented: the basic design philosophy and computational approach are described. Algorithms used to implement synthesis of the innovative pathological voice model parameters described in analysis in Chapter 2 are then described.

4.2.1 Functional Overview

In contrast to the real-time synthesizer, the software synthesizer operates offline in what used to be called “batch” mode. Each time synthesis is invoked, all control parameters are loaded and then computation is initiated and runs asynchronously to completion (generation of the complete output time series). The resulting output time series is stored in memory; the user then initiates audio output to the loudspeakers via the PC's sound card adapter (which has MATLAB drivers). Offline generation permits non-causal (out of time sequence) signal processing techniques, which eases the process of synthesis by permitting various segments of the output to be generated in computationally convenient stages rather than in the strict output sample time sequence. For example, an AGC (automatic gain control) is not necessary, because the whole output time series can be scanned for its maximum and scaled accordingly before it is output to the D/A. Also, different components of the result such as a complete one second periodic LF source time series and a complete spectrally-shaped aspiration noise time series can be completely calculated in different modules and later recombined in the appropriate ratio depending on the desired NSR (noise to signal ratio).

An overview summarizing the major features of the software synthesizer is displayed in Fig. 4.8. After invocation, the synthesizer inputs a set of user-specified control parameters defining the first synthetic vowel to generate; initializations are performed, and the GUI (graphical user interface) is started. The first computation of the output time series then performed; as shown, the major portion of this effort is the generation of the source time series, which may incorporate nonperiodic features such as high and low frequency AM and FM and aspiration noise. Computation of the output time series may take from 1 second to minutes depending on the features enabled. In its normal operation state, the synthesizer waits for user keystrokes to invoke various tasks, such as modifying a model parameter, generating audio output of the synthetic vowel, or loading/storing parameters or time series to files. The three main GUI's of the synthesizer are shown in Fig. 4.9a-c. The primary screen (Fig. 4.9a) allows the user to access sliders and buttons to control I/O, invoke playbacks, and vary model parameters controlling fundamental frequency, power, and the nonperiodic features. Two sub-screens allow interactive modification of LF parameters (Fig. 4.9b) and formants (Fig.4.9c).

4.3 Summary

The function and implementation of two synthesizers has been explained. A hardware-based real-time synthesizer was constructed with the capability of generating immediate responses to changes in voice model parameters. The real-time synthesizer was designed with an expansible architecture allowing easy addition of new features and model parameters, deterministic real-time performance, and the capability to perform real-time closed loop control by performing all system computations within a single sample period. A software synthesizer based on MATLAB was implemented to provide the capability to quickly code and evaluate complex algorithms.