Software Emulation of a Hardware Voice Synthesiser

Pawel Wozniak, University of Huddersfield

pawel.a.wozniak@gmail.com

Accepted date: 17th December 2018

Published date: 13th March 2019

Abstract

The aim of this project was to develop a fully functional emulator of the Speech Plus CallText 5010 hardware voice synthesiser used by Professor Stephen Hawking. Successful completion of the project would allow him to preserve his voice and would greatly reduce the complexity of the communication system he had been using. There were only two fully working hardware boards in existence, and these were already showing major signs of wear. The goal was to retain the exact characteristics of the voice and all the functionality of the original board. It was achieved by reverse engineering the Digital Signal Processor chip present on it, developing an emulator of the chip and merging it with an already existing custom-made CPU (Central Processing Unit) emulator. The operation of both emulators was carefully verified and validated at all stages of development by comparing it with hardware and making sure that the results are bit-perfect. The origin of the project dates back to 2010 and the final result is a collective effort by a number of people. There have been numerous attempts to copy the behaviour of the synthesiser in the past; however, the emulator created as a part of this project was accepted for use by Professor Hawking.

More technical details of the project and audio samples can be found at

www.pawozniak.com

Keywords

Pawel Wozniak, Intel Corporation, Stephen Hawking, electronics, engineering, computer systems, software emulator, hardware voice synthesiser

Acknowledgements

I would like to give special thanks to all the people involved in the project, namely: Peter Benie, Jonathan Wood, Sam Blackburn, Jon Peatfield, Eric Dorsey, Patti Price and Mark Green. The project has been a great challenge that has lasted many years. It would not have been possible to complete it without everyone’s teamwork and passion.

I am also very grateful to my team at Intel Corporation: Stephen Baldwin, Wieslawa Litke, Ali Aram and Martin Tschache, and my mentors at the University of Huddersfield: Pavlos Lazaridis and Violeta Holmes, for all the invaluable knowledge, support and guidance they have provided me with during my years of education.

The history of Professor Hawking's synthesiser was based on an unpublished paper provided by Jonathan Wood, Professor Hawking's Graduate Assistant with his permission.

List of abbreviations

CPU – Central Processing Unit

DAC – Digital-to-Analogue Converter

DSP – Digital Signal Processor

RAM – Random Access Memory

ROM – Read-Only Memory

Introduction

The Speech Plus CallText 5010 Hardware Voice Synthesiser has been used by Professor Stephen Hawking as a primary way of communication since 1985. The hardware was made back in the 1980s and, due to its age, it became obsolete, fragile, and likely to break. There was an urgent need for a backup in case it failed. The system took up a lot of physical space and its power consumption was relatively high. The aim of this project was to create a software-emulated version of the synthesiser that would make it easier to develop additional functions or adjustments, and significantly reduce the complexity of the entire communication system. There have been a few attempts to provide a software-based solution in the past but they have been unsuccessful. The developed programs were not accepted by Professor Hawking due to the differences in voice characteristics. Therefore, a crucial requirement for the emulator was that it copied the exact behaviour of the original circuit. A low-level hardware emulation approach was chosen as the best solution after considering other alternatives such as copying the circuit logic onto an FPGA (field-programmable gate array) chip, modifying an existing software-based synthesiser, or using Machine Learning algorithms.

The documentation regarding the synthesiser was very limited. At the beginning of the project, Sam Blackburn, Jon Peatfield and Peter Benie had reverse engineered the hardware boards by following the track layout and analysing the program ROM. After months of hard work, they got access to a schematic and peripheral & programmer's manual. There was still no official disassembler, programmer, or simulator available for the Digital Signal Processor (DSP) chip, and the original source code was lost. The work was based solely on the hardware boards, a DSP datasheet and previously

developed disassembler and an Arduino-based DSP programmer.

The project objectives were to:

- reverse engineer the hardware components and the DSP present on the board,
- extract the contents of data and instruction ROM of the DSP,
- verify the disassembler and disassemble the DSP object code,
- develop an emulator for the NEC 77P20 DSP,
- merge the emulator with an existing CPU emulator written by Peter Benie,
- validate both emulators at all stages of development by comparing them to the hardware.

History of the synthesiser

In 1962, Professor John Linvill conceived the Optacon system to help his blind daughter read ordinary print. The development was led by James Bliss and teams at Stanford University and Stanford Research Institute (Linvill & Bliss, 1966)**.** Following a successful demonstration of the Optacon prototype, Telesensory Systems Inc (TSI) was founded in Palo Alto, California, in 1970. TSI focused on developing a line of products for the visually impaired**.**

In 1975 James Bliss spoke with Jonathan Allen, and that lead to the development of algorithms from the Natural Language Programming Group at MIT (Massachusetts Institute of Technology) lead by Jonathan Allen (letter to phoneme algorithms) and from the MIT Speech Communications Group lead by Dennis Klatt (phoneme to speech algorithms) that were licensed by MIT to TSI. By 1980, TSI had produced five working prototypes of a real-time system on custom LSI VTM (Large-Scale Integration Vocal

Tract Model) chips**.**

In 1982, Dennis Klatt developed Klattalk (Klatt, 1987) a real-time lab-based text-to-speech (TTS) system. The voices used in his system were based on his family, and the voice called ‘Perfect Paul’ used in the DECtalk system was based on his own**.**

In 1982, a new company called Speech Plus Inc. was formed and developed multiple text-to-speech systems based on the TSI speech technologies. One of their first products, Speech Plus CallText 5000 Telephone/Voice Module retailed at $2700. This synthesiser utilises a method called the formant speech synthesis. Instead of using pre-recorded voice samples, the voice is generated on the go. The speech output is created using additive synthesis and an acoustic model of the vocal tracts. Parameters such as fundamental frequency, voicing and noise levels are varied over time to create a speech waveform (also called rules-based synthesis). The speech synthesis model was based on the original voice of Dennis Klatt, who made measurements of his vocal tracts, his phonetics and phonology, and on his duration rules. (Klatt, 1979)**.**

During an email discussion on the history of Speech Plus with Eric Dorsey, the software engineer who created intonation algorithms for the CallText boards, Eric mentioned that:

> Speech Plus modified the prosody, duration and thousands of the phonological rules that drove the selection of the formants, bandwidths and amplitudes of the various phonemes. In particular English has 44 phonemes and their formants, bandwidths and amplitudes depend greatly on the phoneme to left and to the phoneme to right of each target phoneme so there are literally thousands of combinations of formants, bandwidths, and amplitudes for each phoneme depending on its context. Speech Plus made thousands of changes to these

contextual rules to improve the intelligibility and naturalness of the voice so, over time, it diverged from the original MITalk 79 voice that Dennis Klatt created. Hence if you listen to Stephen's voice and MITalk 79 there are a lot of differences but they both have their genesis in Dennis Klatt's voice.

Synthesisers using the formant speech synthesis technology tend to sound robotic, but formant synthesiser programs take less memory and resources. That makes them suitable for embedded systems. Another advantage is that they allow complete control over different aspects of the voice, such as prosodies and intonation (Klatt, 1979)**.**

In 1985, Professor Hawking contracted pneumonia during his summer visit to CERN. His condition was life-threatening, and his wife Jane was asked to terminate the life support. She refused and, in consequence, he was subjected to a tracheotomy and lost his ability to speak.

In 1986, he started using a voice synthesiser and adopted the voice as we know it today. The synthesiser he used was a Speech Plus CallText 5000 board that was donated to him by Walt Waltosz from Words Plus. Professor Hawking has also tried using voice synthesisers by DECtalk and Votrax, but much preferred the Speech Plus one. In 1988, he was given a new version of the board, the CallText 5010, but didn't like it as much as the older model because of differences in the voice. He has also said 'I used to be able to dial and answer calls but I can't now'. That was solved by replacing ROM chips in the 5010 board with the ones containing firmware from CallText 5000.

Project background

With time, the hardware synthesiser started becoming obsolete and difficult to maintain. Sam Blackburn, who was a Graduate Assistant to Professor Hawking between 2006

and 2012, started looking into different ways of preserving the voice. An approach using Machine Learning was attempted by Phonetic Arts and the resulting voice was very close to the original, but not close enough to be accepted by Professor Hawking. Another attempt by Intel and Ed Bruckert was to modify a DECtalk board, but again the resulting voice was not accepted. Both approaches had failed to keep to the original voice characteristics such as pitch, break timing, pronunciation, or intonation**.**

After years of unsuccessful attempts, yet another concept was pursued. Eric Dorsey got in touch with Nuance, a company holding rights to the synthesiser that they had indirectly acquired from Speech Plus. Engineers at Nuance found the upgraded source code from the 1996 version of the CallText voice. After a few months of work, they managed to get it very close to the 1986 version of the voice. They recorded samples and sent them to Professor Hawking for evaluation. The match was very close but not perfect, and once again he did not accept it**.**

The low-level software emulation approach was started in 2011 by Sam Blackburn, Peter Benie, and Jon Peatfield. There were only two boards available to them at the time: one used by Professor Hawking and a spare backup. There was no documentation available for the hardware, so they made a significant effort in reverse-engineering the backup board. It was not an easy task, since both boards had to be available at any time and experimenting with the backup posed a high risk of destroying it. They eventually identified most of the circuit components, the two main ones being an Intel 80188 CPU and a NEC 7720 DSP. Peter Benie wrote an emulator for the CPU from scratch but did not validate it at the time. Sam Blackburn managed to extract ROM contents of the DSP using a custom-built programmer, and Jon Peatfield wrote a

disassembler for the DSP machine code. Jon understood most of the DSP code and used an existing emulator to execute it but got stuck when he found an illegal instruction in the ROM contents. The project dissolved in 2012 when Jon sadly passed away**.**

During my placement year at Intel, I provided technical support to Professor Hawking. I repaired some of his communication equipment, such as the ‘blink’ sensor used to detect the movements of his cheek muscle, an audio amplifier, and a spare backup synthesiser board. This allowed me to understand the design of the system and notice how fragile it was. I thought it would be a good idea to emulate the synthesiser board in software. After discussing the idea with Professor Hawking’s Graduate Assistant, Jonathan Wood, I was told that it was attempted in the past. I was given access to the files related to the project and saw how much work had already been put into it. I couldn’t understand why it had been abandoned. Jonathan put me in touch with Peter Benie, from whom I heard more about the project background and learned that the reason was not a technical issue. Peter and I decided to bring the project back to life. Shortly after, I was told that there were still two faulty CallText boards in existence, one of them owned by Intel and used for the development of ACAT software. I repaired another unit, making a total of four working and one broken synthesiser boards, and asked to borrow one of them for the duration of the project.

The system

The system used by Professor Hawking consisted of:

- Permobil F3 wheelchair provided by Permobil,
- Lenovo Yoga 260 laptop running Windows 10 provided by Lenovo,
- ACAT interface software developed by Intel,
- A blink sensor,
- CallText 5010 speech synthesiser with an Intel chassis,
- Speakers and amplifiers developed by Sound Research.

ACAT

Assistive Context-Aware Toolkit (ACAT) is an open-source software developed by Intel to enable people with disabilities to have full control over the computer. It enables users to easily communicate with others through keyboard simulation, word prediction and speech synthesis. Users can perform a range of tasks such as editing, managing documents, navigating the Web and accessing emails.

Figure 1

Professor Hawking editing text with ACAT software and a blink sensor

Blink sensors

Between 1986 and 2005, Professor Hawking had been using a hand switch. It acted as a one-button keyboard and, with the help of ACAT software, allowed him to control everything on the computer screen. However, as his condition progressed, he found it difficult to control his hand movements so a search for a new solution began.

Subsequently Professor Hawking started using a Words+ infrared sensor that was commercially available and allowed him to control the computer with his right cheek. However, it had a problem compensating for external light sources because the infrared LED (IR LED) was constantly on. In 2007 Sam Blackburn, his graduate assistant at the time, developed a ‘blink’ sensor that was a big improvement to the design. It solved the previously mentioned issue in a clever way by feeding a square wave into the IR LED (rapidly switching it on and off) and then applying a band-pass filter. It filtered the output signal from unwanted high and low frequencies that were not related to muscle movement. Years later, Mark Green from Intel Corporation further improved the design by simplifying it and making it more efficient. He also created a few backup devices and a schematic. However, the backups felt too sensitive for Professor Hawking, even after the adjustment.

In 2016 I joined the effort and worked on reverse engineering the circuit. I created a schematic of the blink sensor from scratch. It turned out that the schematic owned by Intel had some errors in the component values and that caused the copies to be significantly different. I used my schematic to fix the sensor backups.

Figure 2

Labelled analogue blink sensor circuit board as used by Professor Hawking in 2017

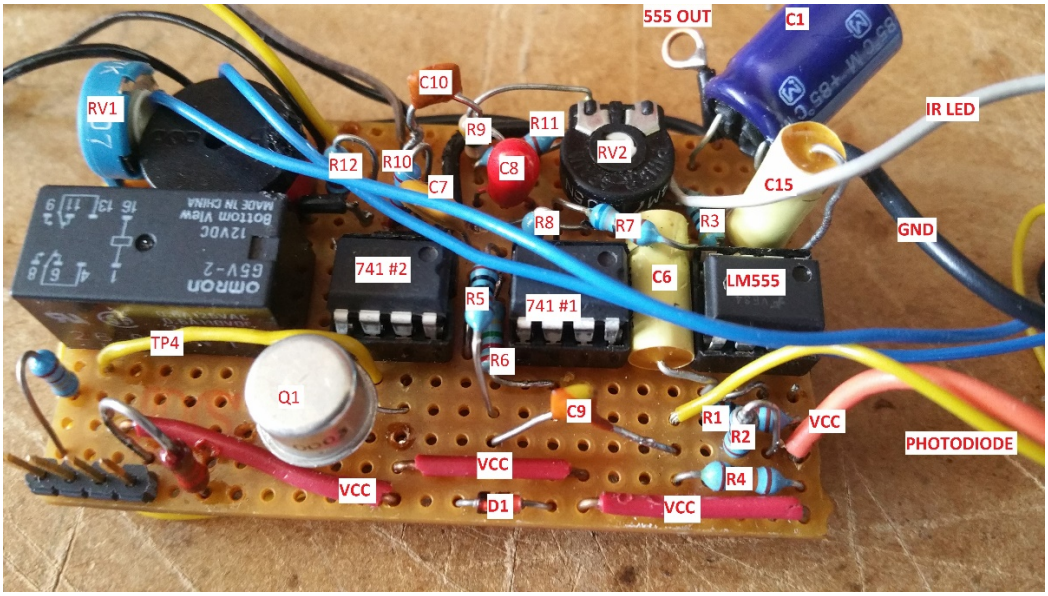
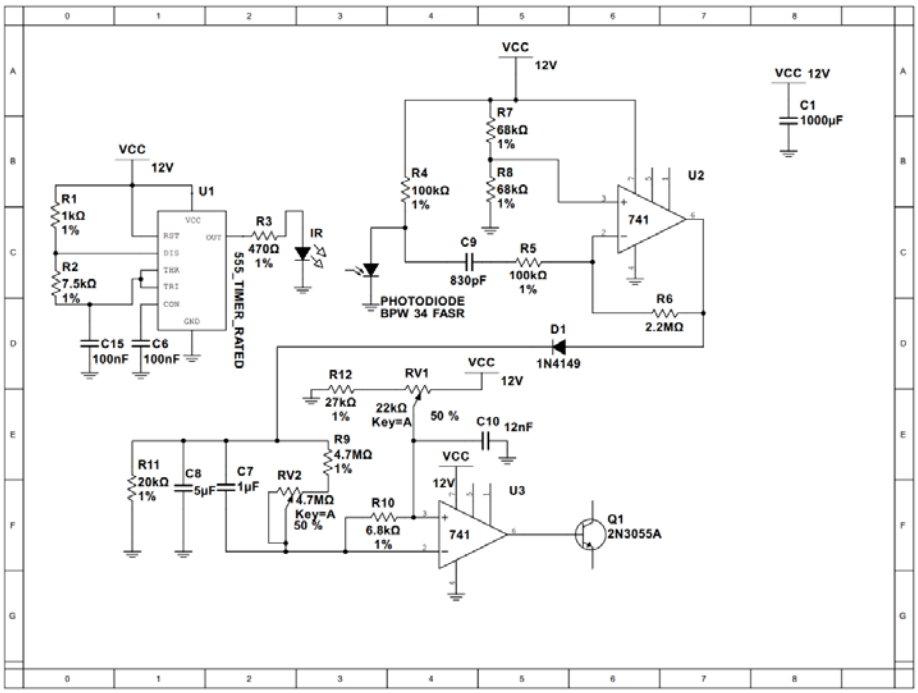


Figure 3

Analogue blink sensor schematic



The blink sensor was based on analogue parts that introduced some inconsistencies and was difficult to calibrate to achieve the desired sensitivity. It had to be adjusted multiple times throughout a day to compensate for various factors.

Intel developed camera-based algorithms to detect facial gestures that are a part of the open source ACAT software and could be used with a webcam. For Professor Hawking it didn't work as well as the blink, in terms of sensitivity and robustness to illumination.

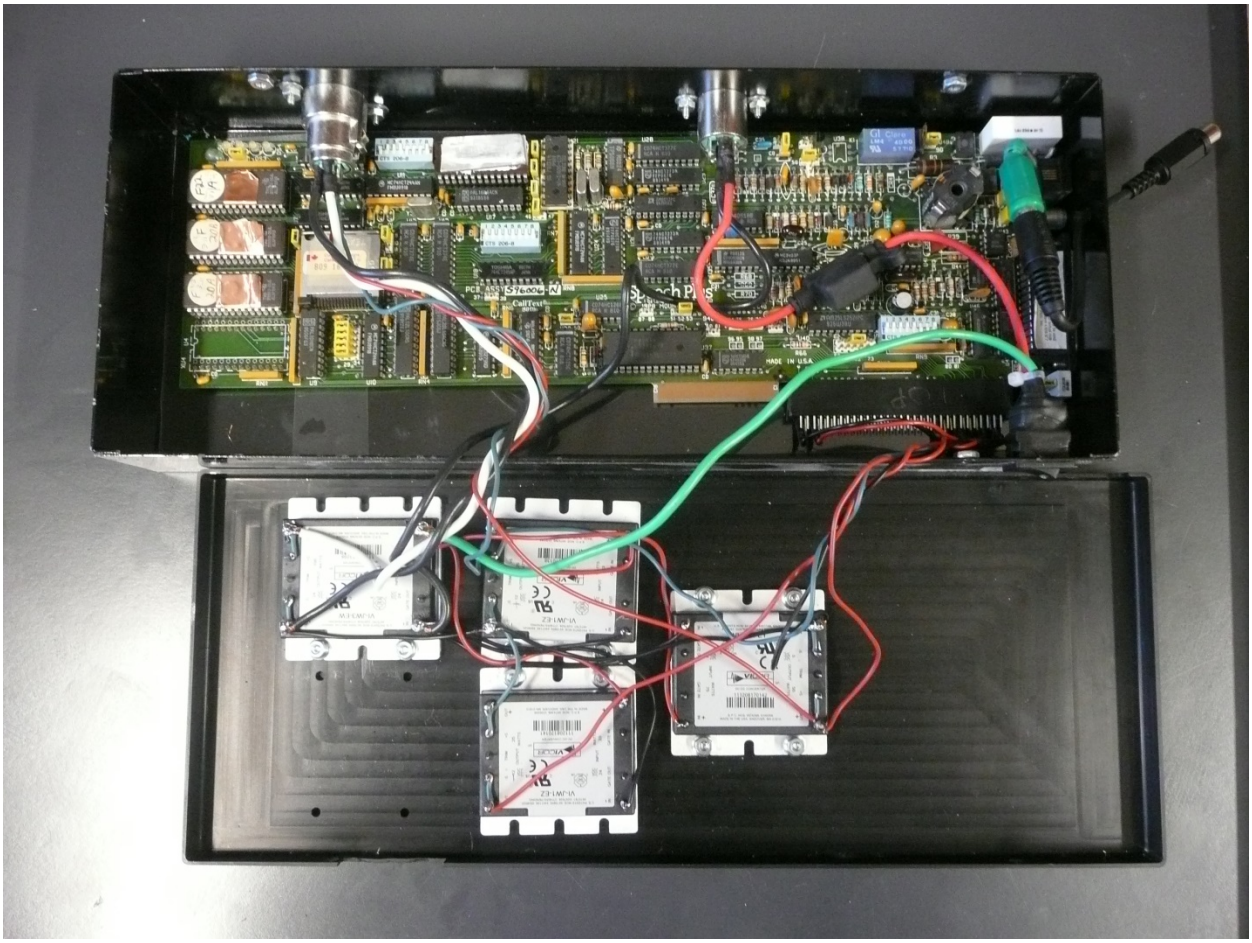
At the end of 2017, the Intel Labs team led by Lama Nachman finished working on a digital version of the blink sensor that was based on a microprocessor sending and receiving infrared waves, so the basic principle of operation was very similar. It was, however, superior to the analogue blink in terms of stability, control, and replicability.

CallText board hardware overview

The CallText synthesiser could be best described as a custom-designed computer system. It was designed for use with IBM PC but another option was to provide it with external power and use it as an entirely standalone device that communicates with a computer via a serial port using a crossover cable. This mode was used by Professor Hawking, so that he could easily send text from his laptop. The resulting voice output could be connected straight to a phone line or to a mono audio device, but only the latter option was used. The voice characteristics such as speed, volume, or pitch can also be regulated by sending escape characters followed by special commands. This feature had to be preserved in the emulator as well.

Figure 4

CallText board with external power supply in an Intel-made chassis (standalone mode)



Central Processing Unit (CPU)

The CallText 5010 board runs on an Intel 80188 x86-16 CPU with three ROM and two RAM chips. The first ROM contains the firmware, the second one some unidentified binary data, and the third one is a library ROM with routines for controlling the hardware. We have tested and proved that the library ROM is not used in the standalone application or the IBM PC mode and can be completely ignored.

The CPU reads English text in ASCII form and processes it using the text-to-phoneme

algorithms developed at MIT. At subsequent coroutines, the phonemes are used to create packets of formants (e.g. bandwidth, frequency, pitch, and amplitude). The packets are passed on to the DSP.

Digital Signal Processor (DSP)

The DSP used in the system is a NEC 77P20 Signal Processing Interface (SPI) chip. It is a prototype version of the NEC 7720 SPI, which made extracting the chip program ROM relatively easy. Unlike the Intel CPU, the DSP can very quickly perform complicated arithmetic operations. One instruction can consist of multiple operations in one cycle.

On the higher level, the DSP acts as a vocoder. It receives five formant packets from the CPU and generates the voice waveforms using vocal tracts model algorithms. There are two routines present in the DSP firmware. The first one receives the data packets from the CPU and writes output data into a queue, and the other one outputs the resulting voice samples to the DAC.

Digital-to-Analogue Converter (DAC)

The AMD AM6012 is a 12-bit multiplying Digital-to-Analogue Converter. It is used to convert the waveform digital sample data to an analogue signal before the amplification stage. While reverse engineering this part of the circuit, we discovered that the two bits of lowest significance had been swapped, probably due to an error in wiring design. The error caused by this is less than 0.05% and would not be perceived by the human ear: $E = \frac{2^2}{2 \times 2^{12}} = 0.048\%$. The error needed to be divided by two because, in half of the cases, when the data is 11 or 00, the output stays the same.

DSP programmer

The DSP programmer was originally developed by Sam Blackburn using an Arduino Mega 2560. He managed to read the contents of the ROM, program a blank chip and successfully generate speech by placing the freshly programmed DSP in the original synthesiser. His intention was to use the programmer as a runtime, but he did not manage to figure out how to perform input/output operations before abandoning the project. The programmer source code came with data and program ROM files extracted from the DSP stored in hexadecimal. We noted that the data ROM contents did not match the instruction ROM addresses, and we found an illegal instruction in the disassembled code. The programmer needed to be validated once again to make sure that the data and instruction files fully matched the ROM contents of the DSP.

DSP ROM contents

The programmer source code was based on a Flash Arduino library for flash-based data collections, written by Mikal Hart under a GNU license. I have carefully compared the program functions against the DSP datasheet to make sure that they meet the chip specification and to become familiar with the ROM files structure.

It turned out that the instruction ROM file, downloaded using the programmer, stored bits in a byte in reverse order (big endian instead of little endian) and had a dummy bit (logic 0) present in every lowest bit of the middle byte. That could be dealt with by the disassembler or interpreter as a part of the emulator.

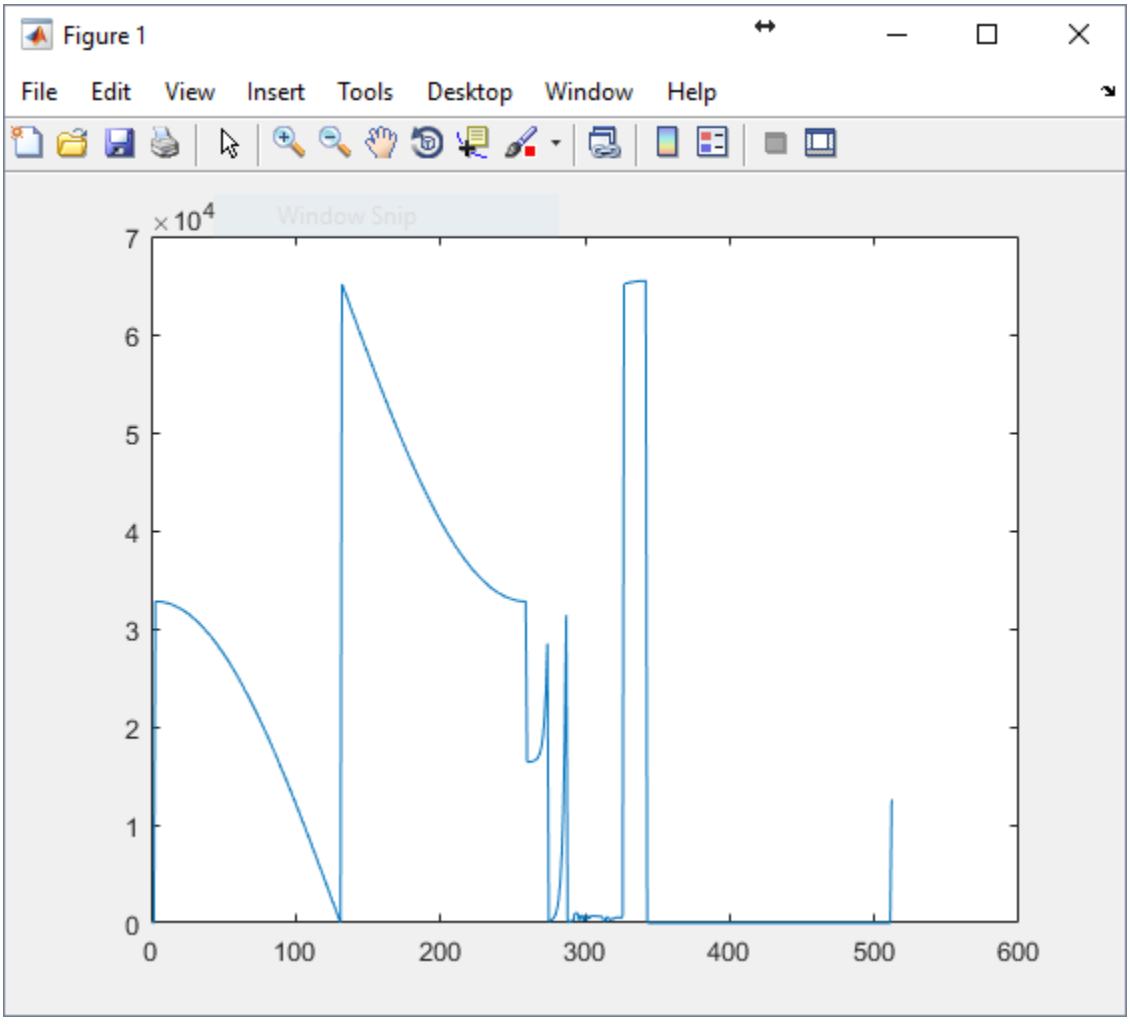
It also turned out that the data ROM was not stored on the PC and Arduino in the same way as it was stored on the hardware DSP. That could lead to difficulties running the object code in an emulator. I wrote a small Python script that reformatted the file to

match the physical structure of the IC using regular expressions and some basic string and list manipulation functions. I then used Matlab to plot the DSP data contents.

Figure 5

DSP data ROM contents plotted in Matlab

```
data = hex2dec(textread('dspdata.txt','%s'))
figure;
plot(data)
```



We could clearly identify a sinusoidal wave, two exponential functions, and some other structures in the data ROM. They are used by the DSP for sample generation.

Data and instruction ROM verification

A brand-new NEC 77P20D DSP chip was programmed using the original ROM dump files. The chip was then inserted into the hardware voice synthesiser to verify that the synthesised voice matches the original and no glitches are present.

Intel 80188 CPU emulator validation

The Intel 80188 CPU emulator is a console application written by Peter Benie. It uses the firmware extracted from onboard ROM to process input strings and provide the data output to the DSP. This part of the emulator must be validated to make sure that this part of the software solution works correctly before developing an emulator for the DSP.

Emulator output data

There are two types of data being sent to the DSP by the CPU. The first is the initialisation data that is only sent once, upon boot, to configure the correct working mode; the second is streamed after the device is given a string of text to synthesise the voice. The CPU emulator initialises with a default input string of ‘Hello. Welcome to the emulator’, printing the data in hexadecimal. Then, it waits for user input.

I adjusted the emulator source code to provide the data in a format better suited for verification and recompiled it. The default input string and any peripheral console printout functions were removed. The DSP write function was modified to print all the data in one line in 1-byte packets separated with spaces, and the order of bytes was changed from big endian to little endian to match the order they appear in on hardware.

I collected multiple datasets from the emulator output: initialisation header (before the emulator asks for input), data generated with different input test strings (e.g. ‘Hello world’, ‘Hello, how are you?’, a comma character itself followed by a dot, and so on).

These datasets should provide enough data for different use cases.

Hardware output data

I used a logic analyser to extract the information from the data bus between the CPU and DSP. The data collected from the hardware synthesiser and CPU emulator had to be brought to the same format to be compared. I wrote another Python script to process all the logic analyser exports and then compared the hardware- and software-generated samples with a help of Beyond Compare.

All the data generated by the emulator was consistent with hardware dumps. The only inconsistency was that the hardware on some occasions generated more data than necessary. A likely explanation for the mismatch at the time was that the response from the DSP was not emulated correctly. It was not possible to fully verify that hypothesis before the DSP emulator was developed.

DSP emulator development

The DSP part was based on higan, an open-source multi-system emulator. The processor emulated in higan is NEC 7725, which is very similar to the 7720. Release notes for the higan suggested that the emulator had been recently updated and should be pretty much bit-perfect. The license under which it was released, GPLv3, allowed us to use and modify the source code.

There are a few differences between the 7725 and 7720 chips. The former has more ROM and RAM memory, one more temporary register, and three additional instructions. Its instructions are 24-bit long, as opposed to 7720's 23 bits. The data words are 16-bits long, while in 7720 they are only 13-bits long. The emulator was developed by extracting the DSP part from higan and translating it to a standalone C program. Then, it

was adapted by changing the sizes of the registers and modifying the instruction interpreter. A check was embedded into the code to notify the user when the program reaches an illegal instruction that we found in the extracted DSP instruction ROM. If that happened, the emulator would output an error message to help us trace it back to the origin.

The DSP emulator was then merged with the CPU emulator. The DSP firmware outputs 0x7F0 to the DAC as a neutral value. It is slightly below half for the 12-bit data (0x800). In consequence, whenever the voice was generated, and the output changed from no input to the DAC (0x800) to the neutral value (0x7F0), pops and clicks could be heard. That problem was also present on the hardware board and was fixed in the emulator by shifting the offset of the DSP output, but some gain had to be sacrificed.

The output from the DSP emulator can be played using computer DAC which also controls the timing of the samples.

Testing

I comprehensively tested the emulated voice by comparing it against the hardware-synthesised voice. At first, I noticed an issue which caused certain words to not be pronounced correctly, i.e. the emulator paused for a split of a second halfway through the sentence. The glitch occurred only with specific words and I realised it was also present on the hardware board I had been using for validation. It was a backup board and it quickly turned out that the behaviour was not present on the original synthesiser. I found that there was one byte of difference in firmware between the boards. It was most likely to have been caused by repeated exposure of the ROM chips to the sunlight. The problem was fixed by updating the firmware in the emulator with the version taken from

the original board.

To get the best possible results, I took digital sample recordings from the hardware, instead of recording the analogue signal. Then, I converted the recorded data to an array of Pulse-Code Modulation (PCM) samples stored in hexadecimal format using a Python script and saved them into a text file which I used to generate a waveform with Matlab.

The voice synthesiser emulator was then used to generate waveforms with the same input strings for comparison. Some samples were found to be slightly misaligned in the time domain. The reason behind this is that the data recorded from the hardware board contained more zero-value samples at the beginning of the file, due to the way the trigger was set on the logic analyser. I used Audacity to remove these extra samples so that the resulting hardware- and software-generated voices could be compared in Matlab Signal Analyzer tool.

Figure 6

Misaligned hardware and software voice samples in Audacity

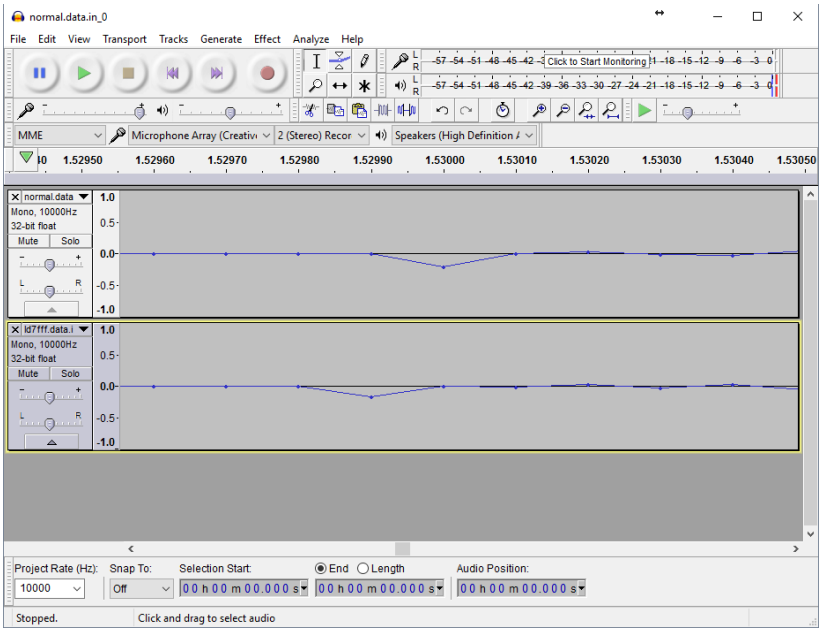
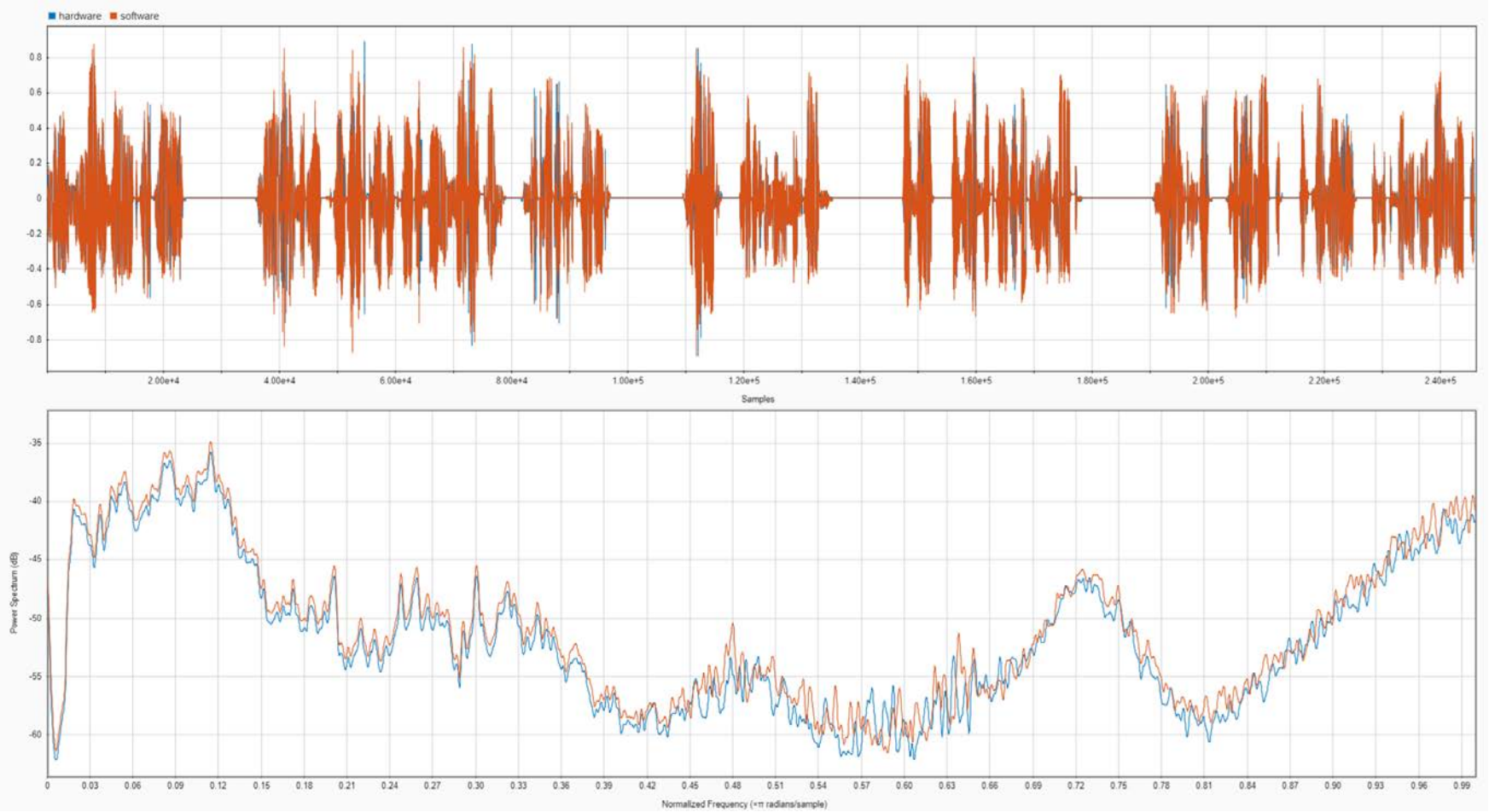


Figure 7

Comparison of hardware- and software-generated voice in the time and frequency domains



In terms of sound duration, the compared signals were an exact match. However, their amplitudes seemed to be slightly different in places, so I investigated further. More samples were taken from both hardware and software and they were always different from each other. Even the samples from hardware did not match perfectly other hardware samples. We noticed that the differences occurred in the fricative sounds, such as ‘z’, ‘f’, ‘v’, or ‘s’. It turned out to be an effect of a pseudo-random number generator implemented on the DSP used for generating these sounds (Klatt, 1979).

In the spectrum domain it was a very close match but small differences were noted. These could also be explained by the effects of the pseudo-random number generator. Another reason may be the fact that the two bits of the lowest significance are swapped on the hardware and this behaviour is not implemented in software. Either way, there was no perceivable difference in the sound. The voice was concluded to be matching the criteria set by the project objectives.

During the testing, the illegal instruction contained in the DSP ROM was never reached by the emulator. Therefore, it did not affect the operation of the program or the voice characteristics.

The last stage of testing was to present the emulator to Professor Hawking himself. It took place in January 2018, using a laptop running Linux OS to drive the emulator that output the audio to the original audio system mounted on the wheelchair. At this point, I had already developed a way of running the emulator under Windows environment, but the solution was a little too messy compared to Linux. The voice turned out to be much cleaner than the original and was the first hardware replacement to be accepted by its owner.

Windows Subsystem for Linux (WSL)

Although the emulator was running natively on Linux, I investigated the possibility of making it work under Windows OS using the Windows Subsystem for Linux environment since that was the system used by Professor Hawking. The biggest issue was the fact that Windows does not officially allow WSL to access any hardware on the host machine. What this means is that native Linux programs can be run under Windows but cannot, for example, play audio. This can be solved using a modified WSL GUI package that runs a pulseaudio stream on the client (Linux) side and installing a pulseaudio server to receive the audio stream on the host (Windows). VSPD (Virtual Serial Port Driver) software was used to emulate the serial connection between ACAT (or, in this case, Putty) and the emulator.

A while later, Peter also developed a native Windows emulator, but it wasn't finished in time; Professor Hawking sadly passed away on 14 March 2018 before he was given a chance to try it.

Conclusions

The project was challenging in several ways. It was not an easy task to find the relevant documentation, reverse engineer the hardware board, develop the low-level software emulator, and verify its operation. Almost eight years after it first started, it was finally brought to a successful end. The emulator was running on a Raspberry Pi 3 with an external USB DAC for additional amplification. Professor Hawking used the emulator as a primary source for his voice for the last two months of his life. When he first tried it, he simply said 'I love it'.

The objectives of the project were met, namely the DSP emulator was developed and merged with the CPU emulator to form a fully software-based voice synthesiser. The voice characteristics were perfectly matched, which has been proved with the

analysis of time and spectrum domains. The voice was accepted by its owner as his own. All the planned work has been carried out although some changes were made to the original project plan. There was no longer a need for developing a solution to inject emulator-generated data packets to the DSP to test the functionality. Instead, samples of the traffic between the CPU and DSP were taken and compared against the software. This solution was less time-consuming and has given more detailed results.

There are multiple advantages of using the emulator over a hardware solution. The whole communication system takes up less physical space and the power consumption is lower. It is easy to keep multiple backups and make changes to the system. The emulator does not break as easily as the hardware does. Upon testing it was also discovered that, even though the voice remained unchanged, the emulator had a clear advantage that was not even considered before: a constant, loud background hiss caused by the degradation of analogue components had gone. Also, popping and clicking sounds generated by the DSP firmware were taken care of, making the voice appear clearer and more defined.

This project was focused on replicating and preserving one of the world's most recognisable voices. Since the voice itself is trademarked, there are no other potential applications for the emulator. This article is an accurate description of the system used by Professor Hawking throughout the years. Hopefully it can help people suffering from similar Motor Neurone Diseases discover and build solutions tailored to their needs.

References

Linvill, J.G., & Bliss, J.C. (1966). A Direct Translation Reading Aid for the Blind. *Proceedings of the IEEE, 54* (1), 40–51.

Klatt, D. (1979). Software for a cascade/parallel formant synthesizer. *The Journal of the Acoustical Society of America, 67* (3), 971–95.

Klatt, D. (1987). Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America, 82* (3), 737–93.