Smart Controller – Artist Talk

Angelo Fraietta

PO Box 859, Hamilton NSW, 2303
email: angelo_f@bigpond.com

Abstract

The Smart Controller is a portable hardware device that allows performers to create music using Programmable Logic Control. The device can be programmed remotely through the use of a patch editor or Workbench, which is an independent computer application that simulates and communicates with the hardware. The Smart Controller responds to input control voltage and MIDI messages, producing output control voltage and MIDI messages (depending upon the patch). The Smart Controller is a stand alone device -- a powerful, reliable, and compact instrument -- capable of reducing the number of electronic modules required, particularly the requirement of a laptop computer, in a live performance. This talk will detail the progress of the project.

1 Introduction

I discussed the Smart Controller at Waveform 2001 (Fraietta 2001), explaining the methodology whereby I was able to develop the software for the Smart Controller using a desktop computer as a hardware simulator. The research, however, had to eventually lead me to the development of a physical hardware device (otherwise the Smart Controller would always remain a theoretical device), which in turn spawned two other products that have commercial potential, enabling the research to partly fund itself before its completion. Brief details of these products will be given later in the document.

The last year has been quite successful in that I have been able to develop a prototype of the Smart Controller in a stand-alone box. The other side of the coin, however, is that this research has been extremely taxing – mentally, emotionally, and spiritually -- in that there were many problems that occurred that made it look as if the project was doomed to failure; however, the people acknowledged at the end of this paper helped to ensure this was not the case.

2 The Resource Model

I had originally intended using a PC-104 embedded PC to run the Smart Controller under RTEMS. I found on the RTEMS newsgroup that a particular PC-104 system had successfully run RTEMS (Wasierski, 2001), and obtained the supplier details. I found the URL describing the device online, however when I jumped to the online catalog, it led me to the DIMMPC. The DIMMPC evaluation board had a PC-104 interface and so I assumed this was the device I required. I bought the evaluation board and DIMMPC (costing approximately $1300 after shipping, customs and GST), and upon opening the box found that I did not get a PC-104 single board computer at all -- the DIMMPC was a whole 386 PC motherboard on a single chip. I had originally intended returning the items, however, after connecting the evaluation board to a monitor and power supply, I found that I was able to run the Smart Controller software on it with no problems at all. This effectively gave me a smaller device, however, I had to design a circuit board and obtain the 144-pin socket to mount the chip. This has now turned out to be a very cost effective and space efficient alternative to the PC-104 system that I had originally intended. Additionally, an advantage of this system is that someone can upgrade to a faster CPU simply by replacing the chip – a viable alternative if I choose to make the Smart Controller perform DSP.

The next stage in the development was the implementation of the MIDI and control voltage I/O. I implemented this using the 16F877 PIC microcontroller. NKA (formally Neil Kilgour and Associates) provided me with parts, development tools, and engineering advice at no cost, thus making this stage of development relatively painless. I designed the circuit board as a separate module to the Smart Controller CPU, thus enabling the board to be used as a standalone CV to MIDI / MIDI to CV controller, without the Smart Controller. I originally offered these on the ACMA post at cost, however, I received a lot of negative feedback, particularly as I did not offer an option to configure the device. I found that I was able to store configuration data within the EEPROM of the device, and as such was able to configure the device using software through the MIDI ports. What has eventuated from this is that I now have a fully configurable CV (control voltage) to MIDI / MIDI to CV controller available for sale at a very competitive price. These devices are now sold internationally via the Internet. This is the first example of the Smart Controller research obtaining a source of funding generated from technology developed before the project's completion. The devices have been designed so they can be easily upgraded to a Smart Controller later.

A multi-object file stream developed for communication with the external Patch Editor became another source of funding. Quikscribe adopted this steaming methodology for their Digital Transcribing, whereby they have created an Intelligent Audio File (IAF).

The .iaf (Intelligent Audio File) provides the Quikscribe Transcription System with a powerful "unique advantage". Rather than just being able to record, edit and transcribe audio files, the .iaf (Intelligent Audio File) provides Quikscribe with the ability to offer a lot of advanced features, not currently available in any other dictation/transcription product. For example, Quikscribe is able to insert Text Attachments, capture Screen Shots and insert File Attachments. It also has a powerful Built-in Database for management purposes. Lastly it can record, edit (Undo or Redo) and compress audio in real-time. (Quikscribe, 2002)

Apart from the financial benefits, this has led to the satisfaction of providing solutions for industries outside of Creative Arts.

The next stage in development was the intercommunication between the PIC I/O card and the 386 Smart Controller card. Communication between the two boards was achieved by using a PLA (programmable logic array), which communicated with the 386 in a parallel data stream, while communicating with PIC using a serial data stream. In developing this area, I found that there were many areas that errors could and did occur, which had to be identified and corrected. The biggest problem, however, was the speed of the data interchange between the two boards. The maximum acceptable interchange between the two boards must be no greater than 320 microseconds as this was the maximum MIDI transfer rate. I was unable, however, to get the rate below 450 microseconds on the DIMMPC, which in turn caused the device to lose MIDI data bytes. I created this situation by sending continuous sysex blocks of 1024 bytes into the device from my PC MIDI output – this caused a MIDI overflow to occur. I ultimately had to reduce the amount of time in the interrupt and the data exchange. Joel Sherrill asked "Aren't you down to the point of counting instructions?" (Sherrill, 2002) I thought that this was some sort of programmer's figure of speech, however, I found out that this was exactly what was required – I had to actually count the number of CPU instructions that were taking place in the exchange. I performed this by stepping through the code in the MPLAB simulator and was able to reduce the time by implementing some methods in the PIC that are normally considered poor programming practice. The first methods I used were implementing global variables instead of passing function parameters, and using "USE_FAST_IO" directives to prevent unnecessary changes to the data direction registers. This proved effective; however, the exchange rate using the DIMMPC was still 370 microseconds – 50 microseconds too slow. The next method I used was actually counting the instructions using the MPLAB simulator and comparing my "C" code with the compiler generated assembly code. Consider the following code fragment, which writes the most significant bit of a variable to a pin of the PIC.

```
output_bit (SPI_PLA_DATA, pla_out_data.flags & 0x80);
```

This fragment took five machine cycles to execute. The following fragment performs the same function, however, only taking three cycles.

```
if (bit_test (pla_out_data.flags, 7))
   {
    output_bit (SPI_PLA_DATA, 1);
   }
   else
   {
   output_bit (SPI_PLA_DATA, 0);
   }
```

This saving of two instructions actually becomes a saving of sixty-four instructions as the code is executed thirty-two times per exchange. The biggest saving, however, was in the omission of "for" loops in the exchange. The following statement causes the code within the loop to be executed eight times.

```
for (byte_num = 0; byte_num < 8; byte_num++)
```

The problem with the code, however, is that it takes ten cycles every time to perform that line of code, which becomes an eighty machine cycle overhead to the loop. This type of iteration is performed four times per exchange, becoming 320 machine cycles of overhead, which translates to eighty microseconds per exchange. I overcame this by placing the code from the block within the "for" loop into inline functions, and literally called them each eight times. These changes enabled the exchange between the two processors to take place in 250 microseconds – well within the required time. This, however, produced another problem – the speed was now too fast in that sometimes the 386 did not sense the interrupt, which in turn caused a lock up when there was no MIDI or CV input at the PIC. This problem was overcome without too much difficulty.

The next problem encountered caused me the greatest distress in the entire project to date. I had the simulator on the Windows machine running well for over a year, however, attempting to run some patches on the RTEMS machine would create access violations that caused the machine to crash if I clicked madly on the Patch Editor. I searched for days, unable to find where the problem in the code could be. The whole point of the simulator was that the majority of the code was identical, and so it would be easy to find the problems by debugging the Windows machine. The problem, however, was that it would not crash on the Windows machine. After three days of intense debugging, I was physically ill from the stress. That night, I cried out in anguish "Lord, I can't find it! It is beyond me. You have to show me where the problem is." The next day I sat down at the computer and started clicking madly on the Windows Patch Editor. Almost immediately, I received a Code Guard error message. Code Guard generated a report to a text file that actually showed me lines of code that had the error. This was miraculous! The error had been in my code for more than a year; however, it did not show itself until that moment. The reason that it occurred so regularly on the RTEMS machine was because the code runs faster in RTEMS on a 40MHz 386 than it does on a Windows 2000 machine running at 1.133GHz. This supports the concept that a machine designed specifically for this purpose would probably be more effective as an instrument than a laptop or desktop computer running software -- such as Max, PD, or Algorithmic Composer -- because the specific machine does not have to waste time performing unnecessary operations such as updating displays, servicing the many tasks that the operating system starts up, etc…, but rather allocates CPU resources only where they are required. I have been able to make the Smart Controller generate a 50 HZ square wave by toggling the digital output every ten milliseconds using a metronome object, while at the same time generating 100 MIDI messages per second out of each of its MIDI output ports. I have measured the digital waveform with an oscilloscope and was surprised to find that the pulses were exactly 10 milliseconds wide – I would not expect to see this level of accuracy using a non-dedicated device.

3 Conclusion

The Smart Controller is now at a stage where it could actually be used in an installation or performance. There are, however, still more features that I would like to add to the hardware device such as non-volatile storage and retrieval of last loaded patch in the case of a power outage at an installation. Additionally, I must now commence work on the patch editor in order to make it run on multiple platforms. I hope to have prototypes of the Smart Controller available for testing by the beginning of 2003, and hopefully, presenting the device at the International Computer Music Conference (ICMC) next year.

4 Acknowledgments

I would like to firstly give thanks to God in that His grace and mercy "delivered me from the paw of the lion"(Holy Bible 1978, I Samuel 17:37) in that it was often not until I broke down and specifically asked for His help did the solutions to the problems become revealed; to Neil Kilgour for generously providing parts, equipment, and technical advice; and to all the people who participate on the RTEMS users group.

References

Fraietta, A. 2001. "Smart Budgeting for a Smart Controller." *Proceedings of Waveform 2001 -- Australasian Computer Music* Association *Conference.* Australasian Computer Music Association, pp. 43-47.

Holy Bible. New International Version. 1978. New York: International Bible Society.

Quikscribe. 2002 "iaf (Intelliogen Audio File." http://www.quikscribe.com.au/PopUps/iaf.htm.

Sherrill, J. 2002. "Re: Interrupts occurrence." RTEMS User Newsgroup Archives, http://www.oarcorp.com/rtems/maillistArchives/rtems-users/2002/march/msg00210.html

Wasierski, R. 2001. "Re: PC-104." RTEMS User Newsgroup Archives, http://www.oarcorp.com/rtems/maillistArchives/rtems-users/2001/august/msg00129.html