National University of Ireland, Galway
Ollscoil na hÉireann, Gaillimh

Department of Electronic Engineering

Advanced MIDI Guitar Effects System

Ronan O'Malley

B.E. Electronic and Computer Engineering

Final Year Project Report - March 2006

Supervisor: Dr Martin Glavin

Abstract

This project describes a unique digital effects system for guitarists, which is suitable for use in live performances. The project will allow a guitar player to easily configure effects in advance from a PC based graphical user interface, a feature that sets this system aside from conventional effects systems that use cumbersome dials, buttons and switches. This feature also allows the professional user to fine-tune specific parameters, inaccessible in traditional multi-effects units. The pedal board device provides real-time adjustment and selection of effects from a custom built effect database. Both means of control adopt the MIDI protocol, and therefore they may be complemented with any MIDI compatible device such as a sequencer or programme changer.

Declaration of Originality

I declare that this thesis is my original work except where stated.

Signature _____

Date _____

Acknowledgements

I would like to express sincere thanks to Dr Martin Glavin. Since first approaching him with the idea of a guitar-based project, he has given immense encouragement. His consistent guidance and motivation throughout the project helped make it a success. I would like to express thanks to my co-supervisor Mr Cillian O'Driscoll, for his positive input and support. Many thanks to the lab technicians Martin Burke, Myles Mehan and Shaun Porter for there invaluable advice and limitless patience. Finally, thanks to my classmates for ideas, constructive criticisms and coffee breaks.

Table of Contents

Table of Figures

1. Introduction

The aim of this project is to research and develop a digital effects system with real-time MIDI (Music Instrument Digital Interface) control and PC configuration for the electric guitar.

Initial stages of research and development aspire to confirm existing effects theory with Matlab, and develop unique algorithms that can be embedded on a DSP board without extended alteration. A set of variable parameters with boundaries is established for each effect to ensure straightforward manipulation for users who have no underlying knowledge of the algorithm, while allowing professional users advanced algorithm configuration.

The project proceeds with development of pedal board for real-time control. The 8051 micro-controller based control system utilises existing MIDI protocol. The DSP system is controlled in two ways using this technology:

- Static on/off data sent from foot-switches, indicating when an effect should be enabled or disabled in real-time.
- Dynamic positional data sent from a variable pedal, this can alter an effect parameter during guitar play and provides a versatile performing environment.

A separate micro-controller is employed to perform switching and parameter change functions to accommodate the possibility of wireless communication between pedal-board and central signal processor. This link is simulated in this project with a wired asynchronous serial link to facilitate MIDI communication.

Effects developed in Matlab are re-coded in embedded C and run on an Analog Devices ADSP-21364 Digital Signal Processor development board. Operating system

and MIDI decoder control layers are implemented on top of this audio processing data layer.

A MIDI connection from a PC to the DSP provides a transmission path for a Java GUI to enable natural and flexible reconfiguration of effects, and storage of set “patches” which can be recalled at the push of a footswitch.

A small amount of further research has been conducted in the area of advanced effect control using an on-guitar lever, joystick or sensor, in the view to apply for a patent.

This project has also been documented in the form of a formal technical paper, suitable for submission to a conference.

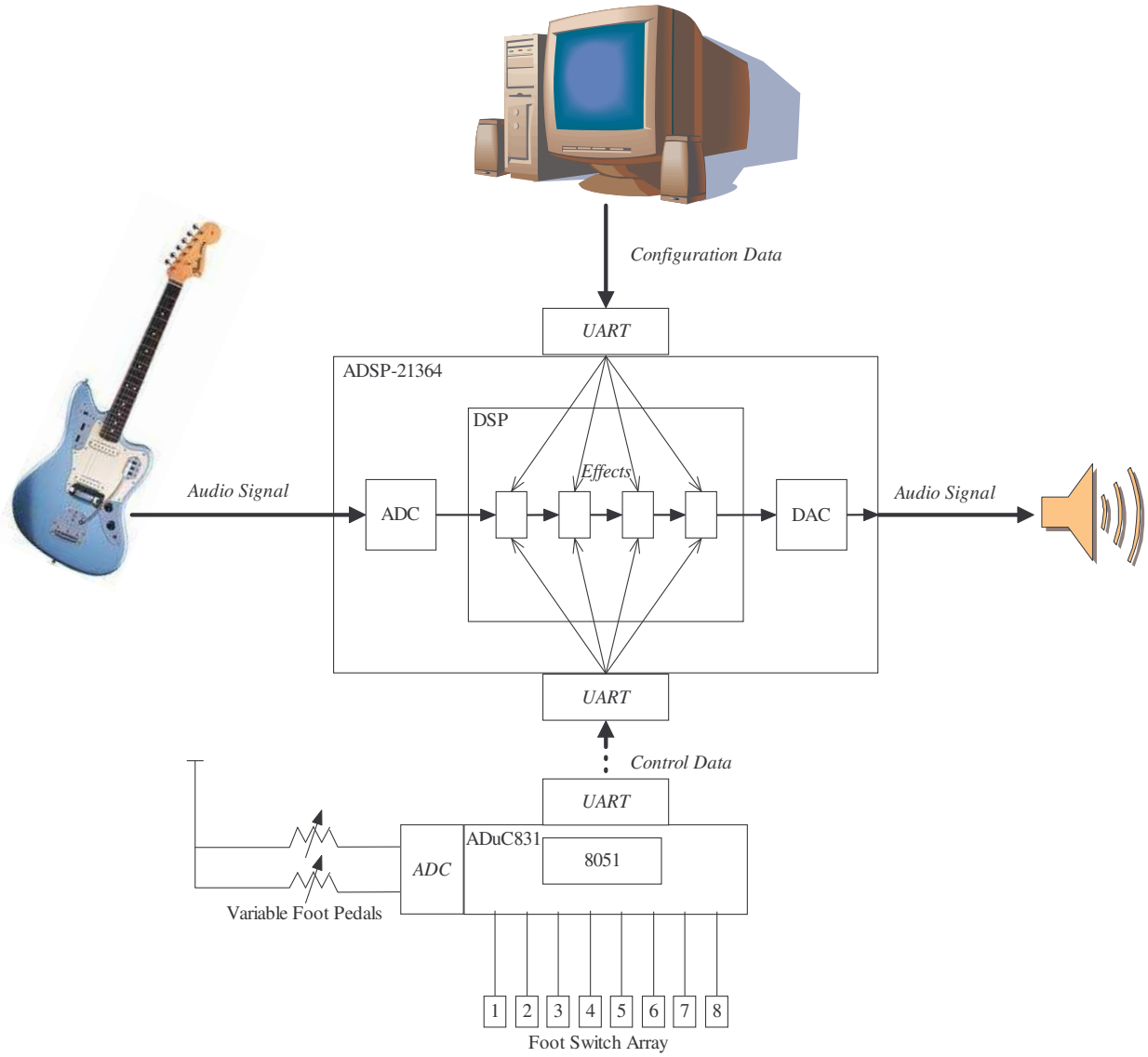

Figure 1. Full System Block Diagram

2. Background Theory

2.1 Guitar Effects

A "guitar effect" is an analogue or digital circuit that audibly modifies the input guitar signal to produce a desired output. There are countless flavours of guitar effects, most of which may be broken down into digital filters, suitable for DSP implementation. Effects can generally be classified into three categories:

- Effects that directly alter the amplitude of the input signal.
- Effects that hold delay's in memory.
- Effects that perform frequency filtering.

Some effects fall squarely into one category, while others have aspects of all three. *Figure 2* shows the typical ordering of effects in a typical signal chain. While effects will function in sequences that differ from this, this order should be adhered to for best results.

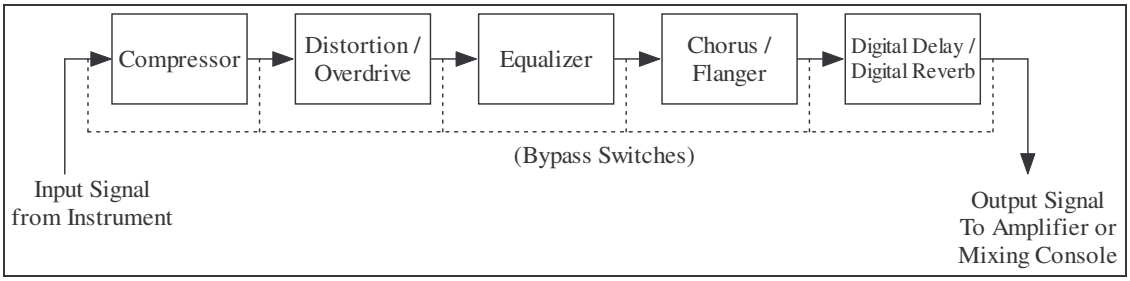

Figure 2. Typical Signal Chain for Multi-Effects Processors [1]

2.2 MIDI Standard

Musical Instrument Digital Interface is the industry standard specification for musical instrument communication and was standardised by the MIDI Manufacturers Association in 1983. The standard defines both the communications protocol and the hardware

interface necessary for transmission and reception of data between musical instruments and musical equipment. A vast database of message functions is available to designers to ensure every possible function is accommodated, and interoperability between equipment is guaranteed.

3. Simulation of Guitar Effects using Matlab.

Guitar effect algorithms were simulated using Matlab. Samples for effect experimentation were recorded in 48 kHz 16 bit wav file format, using a Peavey Raptor electric guitar and Audacity open-source recording software. The Matlab programs were executed on the wav files, and results observed. The output sound files are available on the accompanying project CD (see *Appendix II)*. A set number of variable parameters with clear boundaries were established for each effect, to aid in the process of embedding them.

Below is an outline of each of the effects.

3.1 Overview of Effects

3.1.1 Fuzz Distortion

This effect is also known as symmetrical clipping or the *Hendrix Effect*. The effect is a harsh distortion effect, any sample above or below a threshold is limited to that amplitude. The signal is then amplified to compensate for the loss of average signal magnitude.

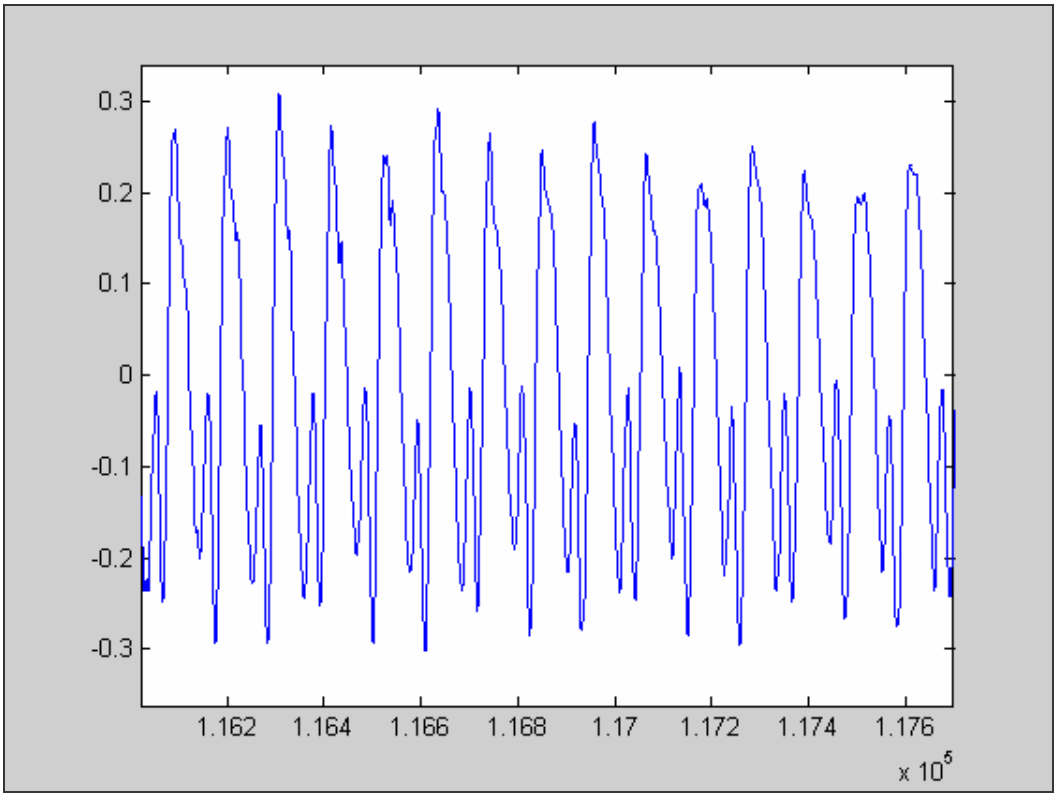

Figure 3. Sound wave before symmetrical clipping is applied

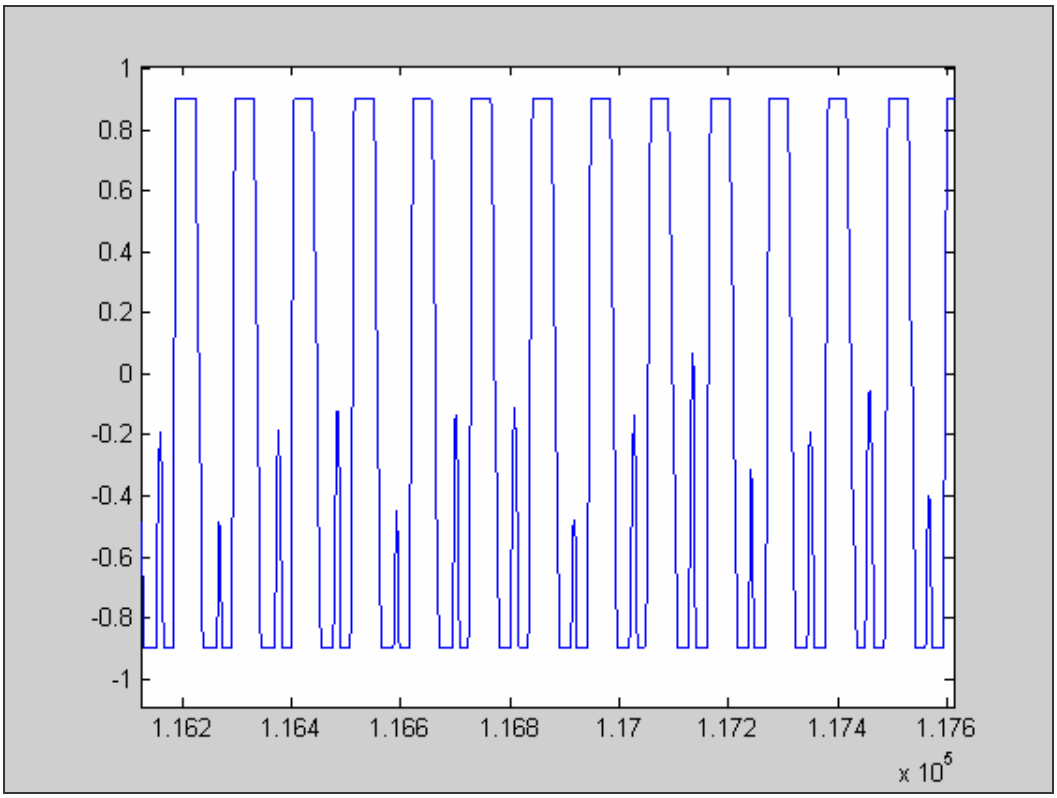

Figure 4. Sound wave after symmetrical clipping is applied

The adjustable parameters for this effect are:

- The clipping threshold.
- The amplification co-efficient.

If different thresholds are used for positive and negative clipping then the effect is asymmetrical clipping. Through experimentation no major audible differences were noted between the two techniques.

3.1.2 Tremolo

A tremolo effect is produced by applying a low frequency oscillating mask to the incoming signal. Triangle-wave oscillator and sine-wave oscillator variants were coded, however the triangle variant produced superior results.

The following parameters and boundaries were established through experimentation with Matlab:

- Frequency of oscillator – 2.4Hz to 24Hz.
- Wave shape of oscillator – Triangle is superior.
- Output may need to be amplified to compensate for overall loss of signal magnitude.

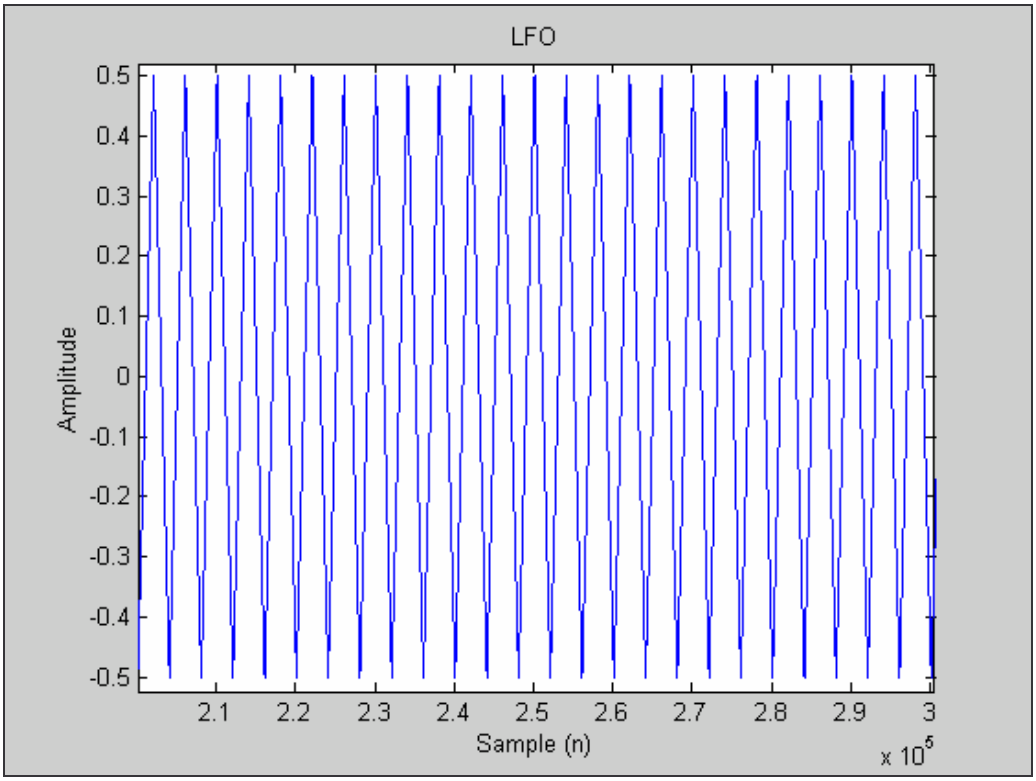

Figure 5. Low frequency oscillator used for tremolo effect

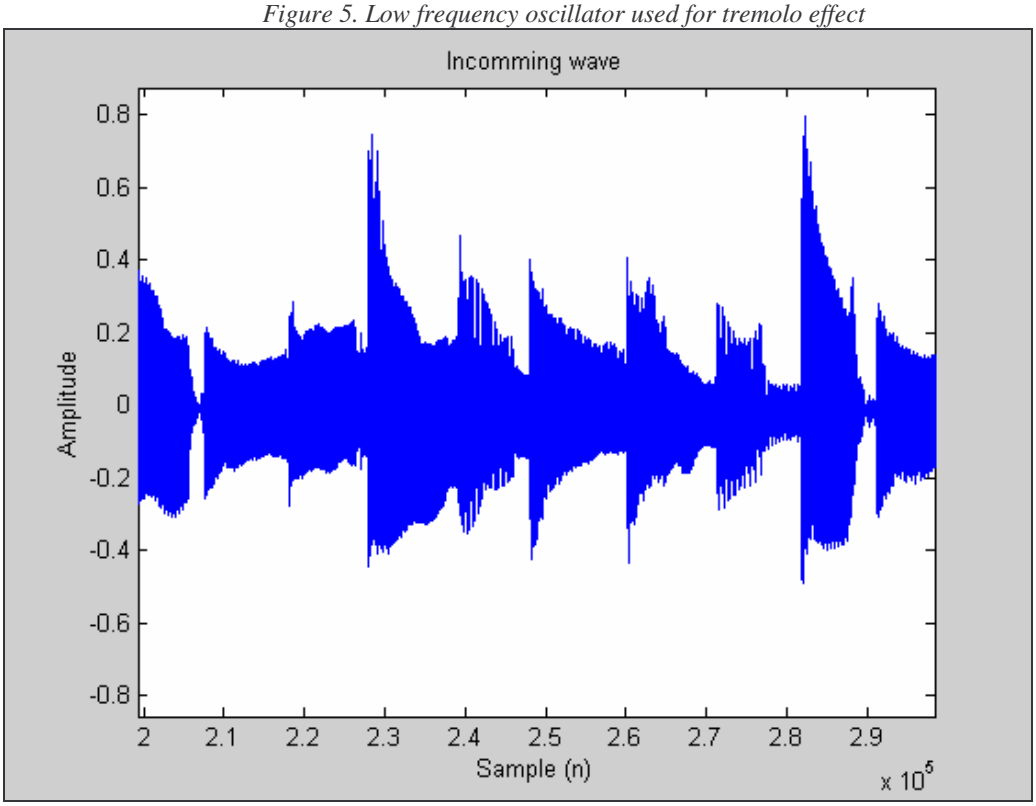

Figure 6. Guitar signal input

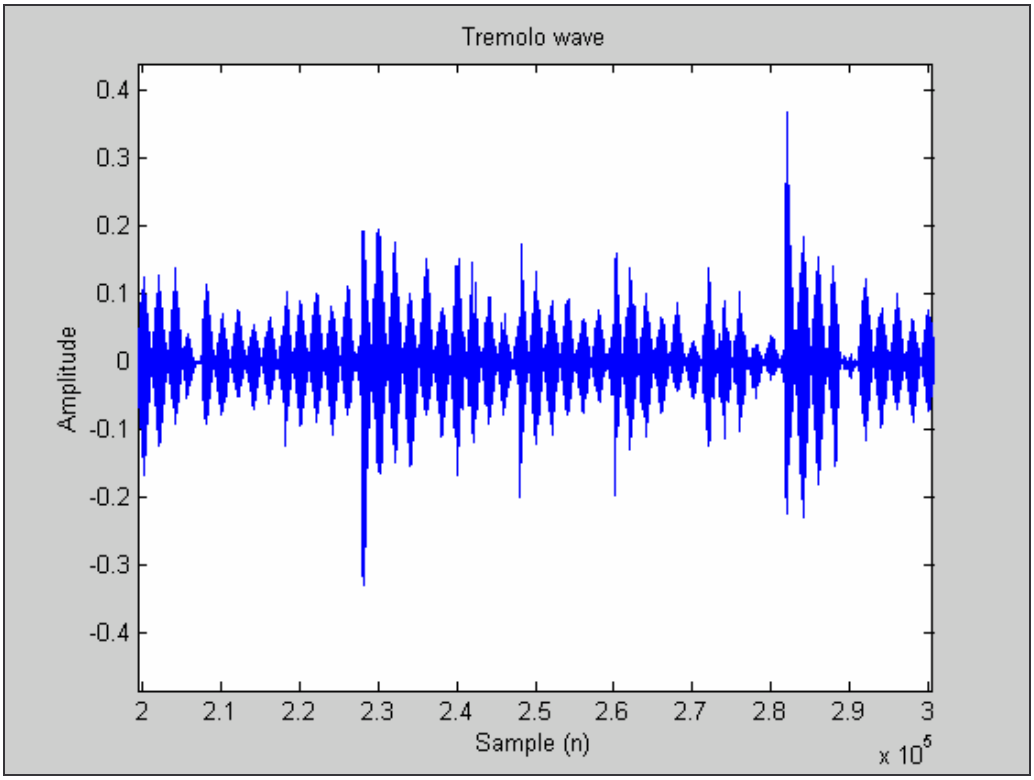

Figure 7. Guitar signal after Tremolo effect applied

3.1.3 Delay

This effect is created by adding delayed, diminished samples to the incoming signal. The following parameters were established to vary the effect:

- Delay period.
- Amplitude of first delay.
- Diminishing rate - this determines the amplitude of the next delay as a function of the current sample.
- No of delays – this is an optional constraint. Generally this would be determined by the diminishing rate reducing the delay amplitude to zero. However this parameter becomes important when embedding this effect.

Varying these parameters can produce a number of different effects. For example, one set of parameter boundaries can produce a traditional *delay* effect; another will produce an *echo* effect.

Delay range (ms)	Modulation	Effect Name
0 – 20	-	Resonator
0 – 15	Sinusoidal	Flanging
10 – 25	Random	Chorus
25 - 50	-	Slapback
>50	-	Echo

Figure 8 Typical Delay Based Effects, DAFX [8], Section 3.3.2

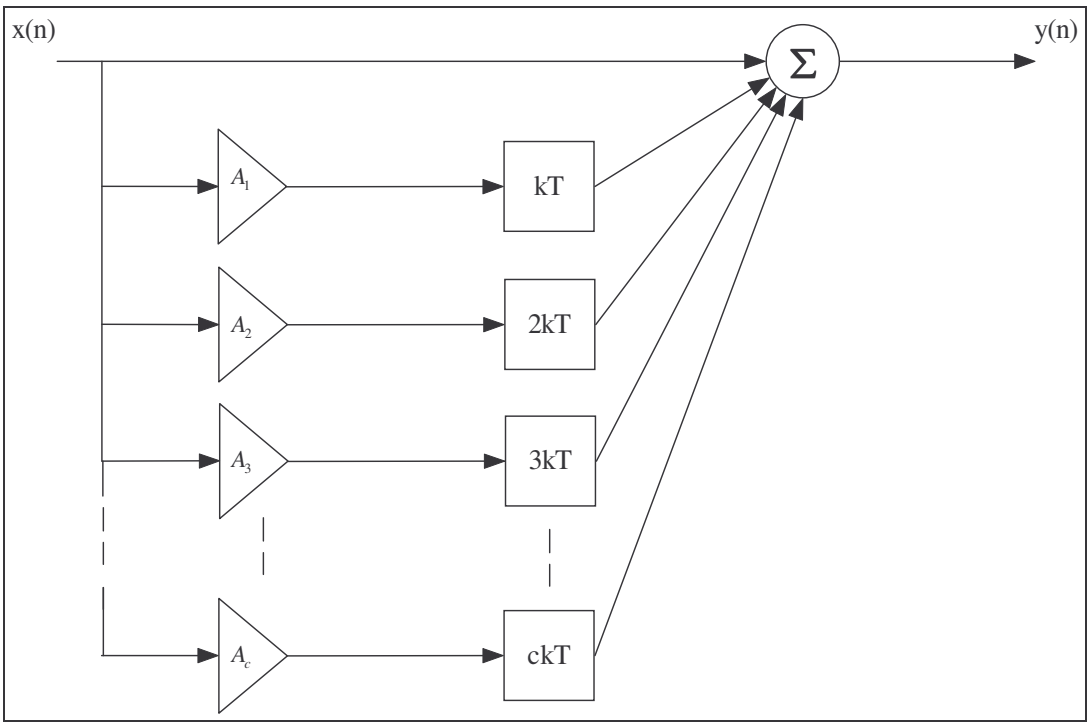

Figure 9 Block Diagram of Delay Effect

$y(n) = x(n) + A_1x(n-k) + A_2x(n-2k) + A_3x(n-3k) +\ldots.+ A_cx(n-ck)$

c is the number of delays.

k is the number of samples between delays. This is a function of the delay period *t* and the sampling frequency *f*.

e.g. For a delay period of 20ms and a sampling frequency of 48 kHz

$t = 20*10^{-3}$

$f = 48000$

$k = t*f = (20*10^{-3})*48000 = 960$ *samples between each delay*

If *k* were not a whole number it could be rounded to the nearest sample, or the value could be interpolated from the surrounding values.

3.1.4 Wah-wah

A "wah-wah" effect was reproduced in Matlab by constructing a peak filter, and oscillating the filter's centre frequency through a set range. A peak filter is a band-pass filter with a narrow pass band and through experimentation, aided by some knowledge of the frequency content of a guitar sound, a parameter range of 500 Hz – 5 kHz was established for the filter's centre frequency. A state variable digital filter from DAFX [2] (*Figure 10.*) was employed to produce the band pass filter, and for simulation purposes the filter's centre frequency was oscillated between it's maximum and minimum bounds

using a triangle wave as a reference. When implemented in hardware, user pedal input will dictate the filter's centre frequency.

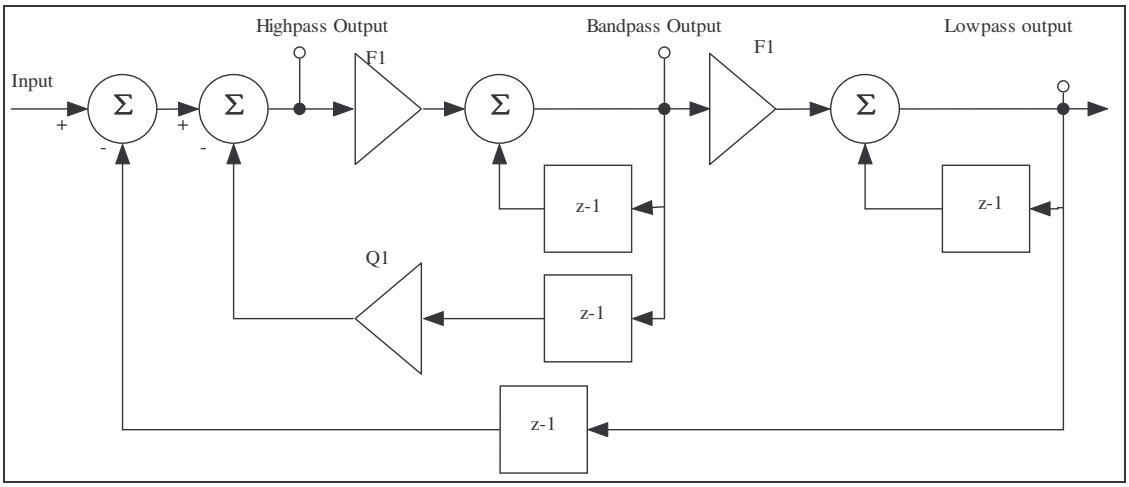

Figure 10. State variable digital filter, DAFX [2], chapter 2

$$y_{lp}(n) = F_1 y_{bp}(n) + y_{hp}(n-1)$$

$$y_{bp}(n) = F_1 y_{hp}(n) + y_{hp}(n-1)$$

$$y_{hp}(n) = F_1 y_{lp}(n) + Q_1 y_{bp}(n-1)$$

$$F_1 = 2\sin\left(\frac{f_c \pi}{f_s}\right)$$

$$Q_1 = 2\varsigma$$

- The variables associated with this effect will be:
- Damping Coefficient (ζ) of the filter.
- Centre Frequency of the filter.
- This is dynamically controlled by a variable foot pedal, creating the "wah" sound that gives the effect its name.

3.1.5 Flanger

A flanger effect is constructed of a single delay, oscillated within a short range at a low frequency. Parameter boundaries for this effect were taken from DAFX [2], Section 3.3.2.

Effect variables:

- Delay range (0 to 3ms)
- Oscillator (Sin wave, triangle wave)
- Oscillating frequency (1 Hz)
- Amplitude of the delay (70% of the input sample)

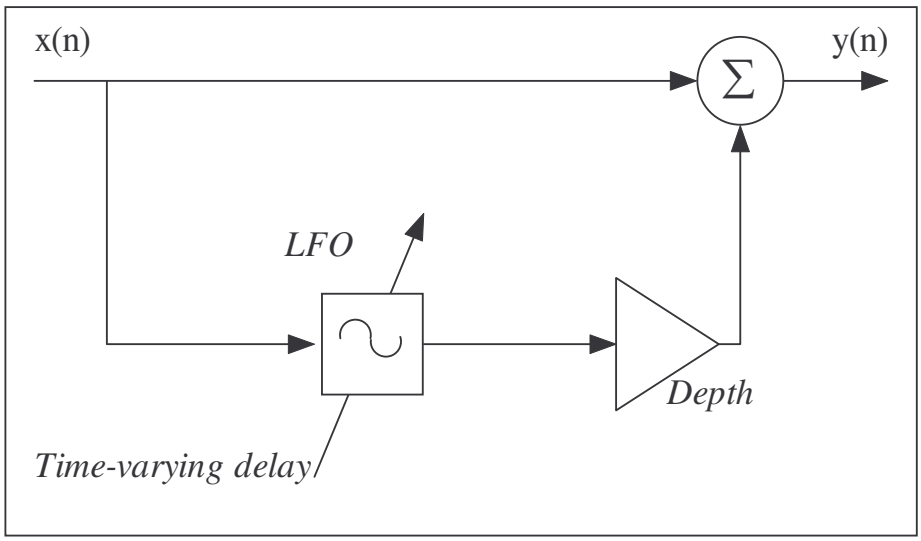

Figure 11. Flanger system block diagram

3.2 Challenges Encountered

Some guitar effects are much simpler to implement using analogue circuitry that digital processing. For this reason, problems were encountered while trying to simulate

the chorus effect. The chorus effect is similar to the flanger effect, except the low frequency oscillator is low frequency random noise rather than a sin wave. The other differences are a much longer delay length, and numerous delays are used instead of the singular delay used in the flanger effect. It is these differences that make this approach to chorus too computationally intensive for digital signal processors to process in real-time. "Modern digital processors produce chorus in a different way, which provides a stronger chorus effect, but also adds a small out-of-tune effect. It is produced by mixing the original signal with one that is modulated slightly flat then sharp, and further delayed a small amount."[3]

4. 8051 Pedal-board

The 8-bit assembly code system was embedded in an 8051 based microcontroller, within an Analog Devices ADuC831 development board. The pedal-board incorporates this microcontroller as it is designed to be a stand-alone entity, which could ultimately also control a wireless transmitter.

4.1 ADuC831 Overview

"The ADuC831 is a fully integrated data acquisition system incorporating a high performance self-calibrating multichannel12-bit ADC, dual 12-bit DACs, and programmable 8-bit MCU on a single chip. The microcontroller core is an 8052, and therefore 8051-instruction-set compatible with 12 core clock periods per machine cycle." [4]

The ADuC831 was chosen for ease of use, on board ADC's and UART.

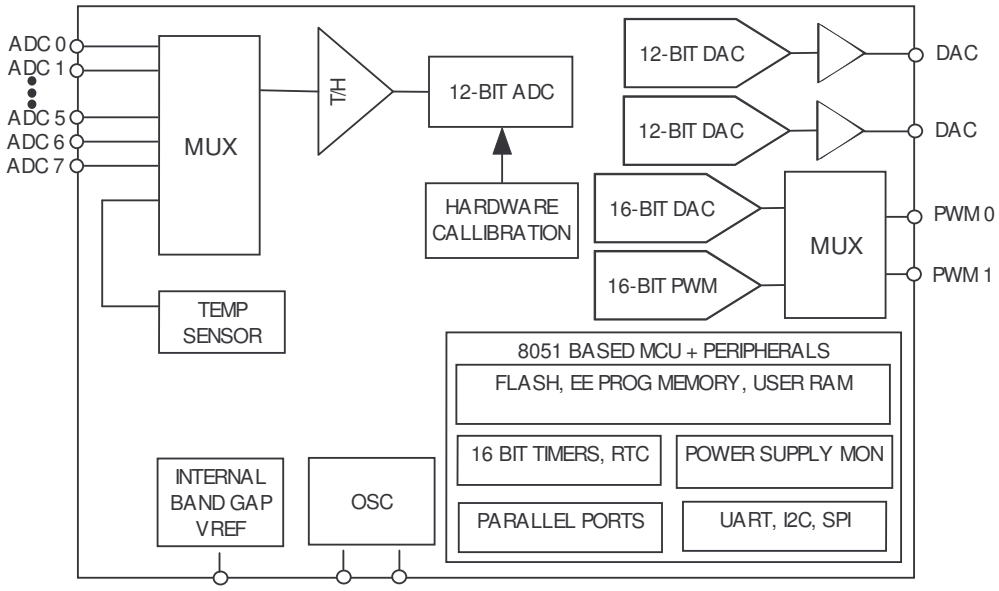

Figure 12. ADuC831 Functional Block Diagram [4]

4.2 Operating System

An operating system was coded to obtain data from the pedal board input equipment, convert the data into MIDI message format and transmit over an asynchronous connection to the DSP unit.

The real-time operating system firstly polls two variable foot pedals and compares the current value with the last transmitted value in memory. If the pedal's value has changed since the previous transmission, then the new value is converted into MIDI format, and three MIDI messages are sent. These messages indicate that:

- there is a control change
- the channel number
- the controller number of the pedal
- the pedals new value

The ADuC831's on board ADCs produce a twelve bit result upon reading the voltage from the on board potentiometer. To conform to MIDI protocol the seven most significant bits of the ADC reading are used to determine the pedals position, which provides a sufficiently accurate resolution.

```
MOV A, ADCDATAH          ; includes the channel number
MOV B, ADCDATAL          ; retrieve LSByte of data

ANL A,#00001111b         ; remove adc channel number
ANL B, #11110000b        ; remove 4 least significant bits
ORL A,B                  ; gather data together
SWAP A                   ; change nibble order
RR A                     ; rotate right to give 0 and 7 msb 0vvvvvvv
ANL A, #01111111b        ; ensure bit 7 is 0 to conform with midi
```

A similar functionality is employed for the switches that control effect on/off state. Memory holds the most recent switch value transmissions. If the value of a switch has changed, three MIDI messages are transmitted corresponding to the switch's new value. The DSP board will receive these messages, decode them and apply the corresponding changes to the embedded effects.

4.3 Stable Reference Voltage

A circuit was constructed to ensure a steady reference voltage from the board. This is linked to two potentiometers which act as variable foot pedals. The steady reference voltage is essential so that when the pedal is still, the data read on the ADC is the same for each consecutive poll. If this were not the case the operating system would interpret the fluctuating readings as a change in pedal position and transmit MIDI messages.

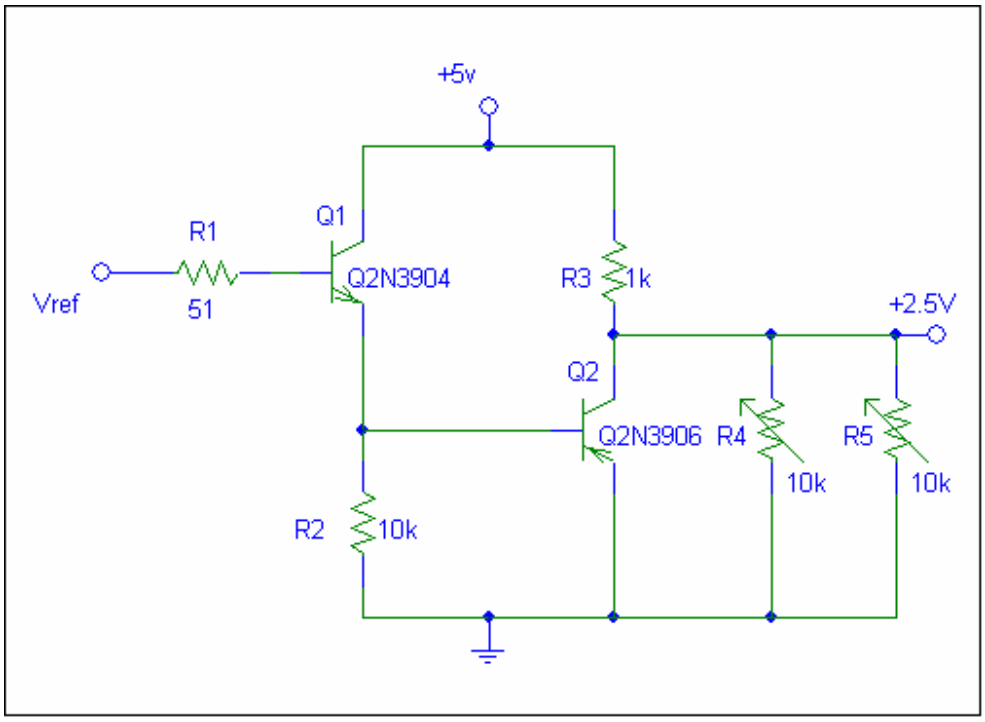

Figure 13. Voltage regulator circuit. [5]

4.4 MIDI messaging

The MIDI protocol's *control change* message format differs from other standard MIDI messages as the function they control is dictated by the musical equipment designer, and is not set out in the standard. It is for this versatility and their relevance that they were chosen to perform system switching functions.

This system could be extended by embedding a 'scene selection' arrangement. This would allow the guitarist to store groups of effect configurations (*patches* or *scenes*), and recall them at the push of a switch. This functionality would use the *program change* midi message format and allow sophisticated effect changes, suitable for live performance. "Program change messages enable the sound of an instrument to be changes from one present sound to another preset sound, rather than a means of adjusting individual parameters."[6]

Function	Midi Message	Definition
Control Change	1011 nnnn	Midi Channel (1 – 16)
	0ccc cccc	Controller Number (0 – 127)
	0vvv vvvv	Control Value (0 – 127)
Program Change	1100 nnnn	Midi Channel (1 - 16)
	0ppp pppp	Program Number (1 – 16)

Figure 14. Midi messages used in pedal-board operating system [7]

MIDI channel	Binary Value	Assignment
1	0000	Variable pedals Controller 0 (0000) – pedal 1 Controller 1 (0001) – pedal 2
16	1111	Foot Switches Controller 0 (0000) – switch 1 Controller 1 (0001) – switch 2 Controller 2 (0010) – switch 3 Controller 3 (0011) – switch 4 Controller 4 (0100) – switch 5 Controller 5 (0101) – switch 6 Controller 6 (0110) – switch 7 Controller 7 (0111) – switch 8

Figure 15. Table of MIDI channel assignments for pedal board system

Examples of MIDI messages transmitted by the system:

Pedal 1 has a new value of 0xC35 = 1100 0011 0101, 7 MSB taken => 110 0001

MIDI message constructed

1011 **0000 –** Control change MIDI channel one (variable pedals).

0**000 0000 –** Controller 0 (pedal 1).

0**110 0001** – New value.

5. Embedded DSP of Effects and Operating System

All guitar effects outlined in section 2.1 were re-written in C and embedded in an Analog Devices SHARC processor. This processor performs the audio signal processing concurrently with asynchronous serial reception, MIDI message decoding in addition to control and parameter change.

5.1 ADSP-21364 EZKIT-LITE and Visual DSP++

The ADSP-21364 is aimed specifically at professional audio processing and is therefore ideal for this task. The development board is well equipped with audio-in and audio-out RCA jacks enabling direct connection of guitar and amplifier using standard audio cables. It is capable of processing twenty-four bit audio samples at up to one hundred and ninety-two kilo-bits per second. The on board audio is transported using Phillip's I^2S protocol, allowing straightforward compatibility with audio peripherals. Similar to I^2C, I^2S (Inter-IC-Sound) is a serial bus interface standard, used for connecting digital audio devices together. The board boasts a USB JTAG interface allowing simple device programming and debugging and comes equipped with six synchronous serial ports, two SPI ports, and a parallel port.

"The ADSP-21364 is a high performance 32-bit/40-bit floating-point processor optimized for professional audio processing. At 333 MHz/2 GFLOPs, with unique audio centric peripherals such as the digital audio interface that includes a high-precision 8-channel asynchronous sample rate converter among others, the ADSP-21364 SHARC processor is ideal for applications that require industry leading equalization, reverberation and other effects processing." [8]

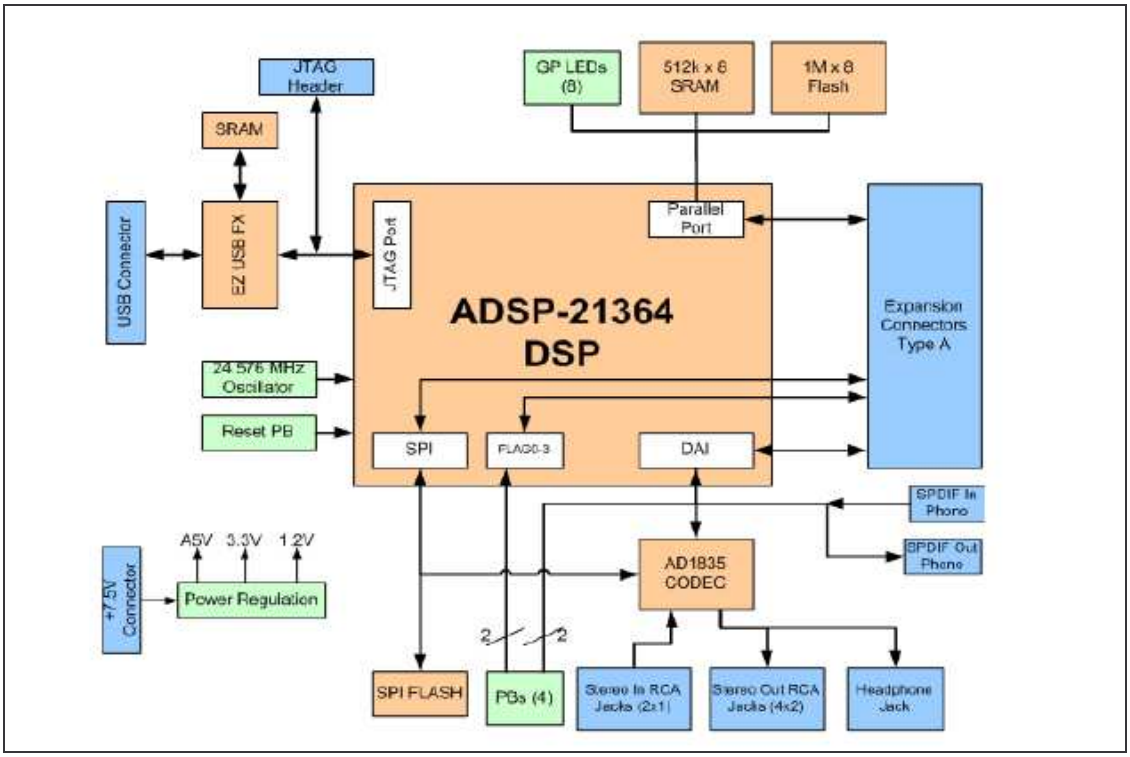

Figure 16. Block diagram of ADSP-21364 Architecture [9]

5.2 Digital Audio Interface and Signal Routing Unit

"The digital audio interface (DAI) is comprised of a group of peripherals and the signal routing unit (SRU). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRU connects the peripherals to a set of pins and to each other, based on a set of configuration registers. This configuration allows the peripherals to be interconnected to suit a wide variety of systems. It also allows the ADSP-2136x processor to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count." [10]

A Visual DSP++ plug-in, 'Expert DAI' was used to configure the signal routing configuration for the EZ-KIT. Several paths must be established in order for the system to operate correctly:

Source	Destination
8051 UART Data	DAI Pin 3
8051 UART Frame Sync	DAI Pin 3
8051 UART Clock	8051 UART Clock
PC MIDI Port UART Data	DAI Pin 2
PC MIDI Port UART Frame Sync	DAI Pin 2
PC MIDI Port UART Clock	PC MIDI Port UART Clock
Serial Port 0 Clock	ADC Clock
Serial Port 0 Frame Sync	ADC Frame Sync
Serial Port 0 Data	ADC Data
ADC Clock	DAC Clock
ADC Frame Sync	DAC Frame Sync
ADC Data	DAC Data
Serial Port 1 Clock	DAC Clock
Serial Port 1 Frame Sync	DAC Frame Sync
Serial Port 1 Data	DAC Data

Figure 17. Table of Guitar Effects System SRU Configuration

5.3 Audio Signal Processing

The first objective in developing the audio processing system was to build a system that sampled audio through the on board ADC and passed it directly out to the DAC. This system is called a *talk-through* system. While a template for a *sample-based* talk-through system existed in assembly language, no C language version existed, so it was undertaken to convert the assembly language version into embedded C aided by a C

version of a block-based talk-through system. Block based systems are suitable for some signal processing applications, such as the fast Fourier transform, but the complex filtering and real-time requirements of this project require sample by sample handling.

Using the EZ-KIT's Signal Routing Unit as outlined in *Section 5.2*, the on board analogue to digital converter was connected to the processor using serial port 0. Each time the ADC samples the incoming audio signal, it triggers the serial port 0 interrupt. Serial port 1 is configured through the SRU to transmit to two of the four on board digital to analogue converters. This allows for output of the processed audio to a guitar amplifier, and a set of headphones.

Upon execution of the serial port 0 interrupt this talk-through system reads the serial port 0 data register for the current sample, and writes it to the transmit register of serial port 1 for output to the DACs. It is between these two tasks that the effect processing takes place.

5.3.1 Fuzz Effect

To implement this effect on the DSP board, the C function *fclipf* was employed.

"*float fclipf (float x, float y);*

The fclipf function returns the first argument if it is less than the absolute value of the second argument, otherwise it returns the absolute value of the second argument if the first is positive, or minus the absolute value if the first argument is negative." [11]

This function fulfilled the same functionality as the fuzz effect coded in Matlab (*Section 3.1.1*).

```
float current_sample;
float clipping_threshold = 38000;

current_sample = *pRXSP0A;          // read serial port 0A
output = fclipf(current_sample, clipping_threshold);

*pTXSP1B = output;                  // write to serial port 1B
```

5.3.2 Tremolo

The triangle wave necessary for implementation of the tremolo effect was reproduced by storing one period of a triangle wave in a circular buffer. The ADSP-21364 contains hardware circular buffers, so there is zero overhead in looping. i.e. there is no need to check if you have reached the end of the memory reserved for the buffer before incrementing the address pointer, as the task of looping back to the first memory location is accomplished through hardware. While the compiler can be intelligent enough to incorporate the hardware circular buffers when assembling C code, to ensure it uses one of them, the function *circptr* is applied to increment the buffer pointer.

"circptr

perform circular buffer operation on a pointer

void circptr(void * ptr, size_t incr, void * base, size_t buflen);*

Description

The *circptr* function is used within a loop in order to implement a circular buffer operation in C/C++. The argument *ptr* represents the pointer that is being used for the circular buffer, *incr* represents the value by which the circular buffer should be

incremented, *base* represents the array on which the circular buffer operates, and *buflen* represents the size of the circular buffer." [11]

Matlab was utilised to output two thousand samples representing a single period of a triangle wave and these were stored in a buffer in the processors internal memory. To perform the guitar effect, the current sample is read in from the ADC, the current triangle wave sample is read from the buffer and the triangle buffer pointer is incremented. The sample is then multiplied by the triangle reference, and output to the DAC.

To control the frequency of the tremolo without changing the size of the buffer, the buffer pointer is updated every *n* interrupts. While this limits the amount of tremolo frequencies available, it is a fast and flexible approach.

```
#define TRI_BUFF_SIZE 2000
int INC_TRI = 4;              // how often to increment pointer
int tremCount = 0;            // count to next pointer update
float tri_buffer[2000] =
{
      0,
      2.000000e-003,
      4.000000e-003,
      6.000000e-003,
      8.000000e-003,
      1.000000e-002,
      …
      …
};
float* triP = tri_buffer;     // pointer to current sample in buffer
```

```
if(tremCount == INC_TRI)
{
      // increment pointer
      triP = circptr(triP,1,tri_buffer,TRI_BUFF_SIZE);
      tremCount = 0;          // reset counter
}
else
{
      tremCount++;
}
// tremolo
output *= (*triP);
```

This counter, linked to the pointer update, results in the following tremolo frequencies for a sampling frequency of 48 kHz.

INC_TRI	Update	F_{LFO} = (Fs/INC_TRI)/2
3	Every 4th interrupt	6 Hz
4	Every 5th interrupt	4.8 Hz
5	Every 6th interrupt	4 Hz
6	Every 7th interrupt	3.4 Hz
7	Every 8th interrupt	3 Hz
8	Every 9th interrupt	2.7 Hz
9	Every 10th interrupt	2.4 Hz

Figure 18. Table of Tremolo frequency settings

For a more precise setting an external timer interrupt could be used to update the pointer.

5.3.3 Delay

It was intended to take full advantage of the hardware circular buffers outlines in *section 5.3.2* when implementing the delay effect. However, this functionality is only possible when the buffer is stored in processor on chip memory. The implication of this was that the buffer could consist of no more than approximately 24000 samples (500 ms), before the processor ran out of memory. As a longer delay was desired, external memory was incorporated into the design. This posed several challenges:

- Hardware circular buffering is unavailable in external memory.
- External memory reads and writes are very slow.
- External memory can only hold 8-bit data.
- Accessing and addressing external memory is far more complex that internal memory.

To overcome the loss of circular buffering a check is performed before each pointer increment to ensure, that if addressing the last sample in a buffer, it circles around to reference the first sample on the next increment.

The system was tested with external memory read and writes within the sample receive ISR. However because of slow access times to external memory, time constraints were unable to be achieved, and most samples were dropped. To conquer this, a small buffer in internal memory is used in the ISR. This small buffer contains only the current input sample and the samples needed for writing the current output. All external memory access was moved outside the ISR. While the processor is idle the small internal memory buffer is filled with the samples necessary to perform the delay effect in the next interrupt and the last input sample is copied out of it to external memory.

External memory is accessed through the parallel port. Specific operation is described in “Analog Devices Engineer to Engineer note EE-220” [12]. Consideration had to taken when considering addressing, and four different memory locations have to be read when accessing a 32-bit word as the external memory is only one byte wide.

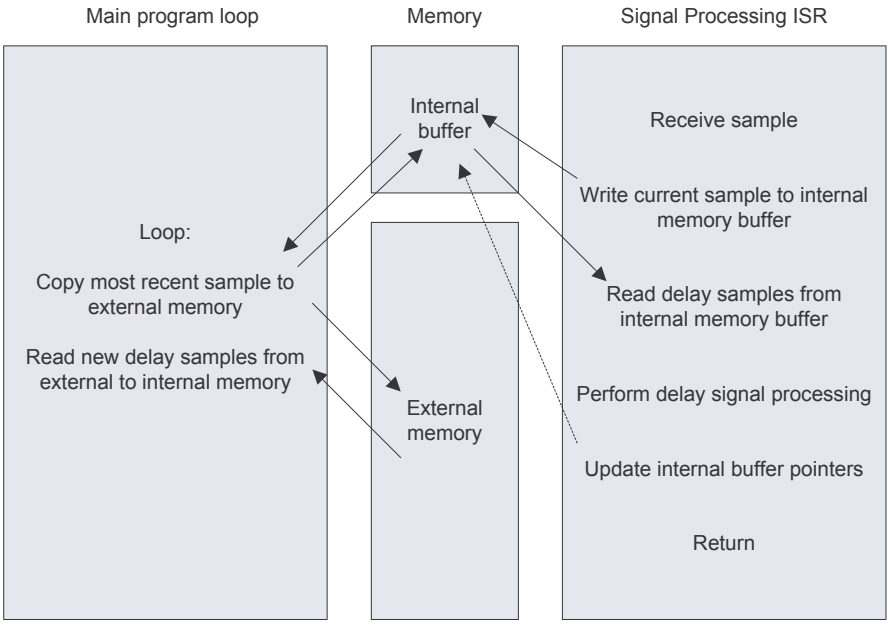

Figure 19. Block diagram of delay memory buffering application.

```
void main()
{
	while(1)
	{
		// write current sample to external memory
		writeToExt(*curSampP,curExtAdd);
		// read delays from external memory
		readFromExt(internalSample, Xdly1);
		readFromExt(internalSample + 1, Xdly2);
		readFromExt(internalSample + 2, Xdly3);
		... ;
```

```
        }
}
void talkthroughISR()
{
        // delay effect
        output = 0.7*current_sample + 0.3*(*dlyPtr1) + 0.25*(*dlyPtr2)
        + 0.2*(*dlyPtr3) + ... ;
}
```

5.3.4 Flanger

A similar approach to that outlined for the tremolo effect in section 5.3.2 was adopted for embedding the flanger effect. Values representing half a period of a sine wave were generated using Matlab, and stored in internal memory. However, it was important to achieve a much higher resolution than with the tremolo triangular wave. For this reason twenty-four thousand sine wave samples were stored in a circular buffer structure, utilising the same *circptr* addressing procedure outlined in section 5.3.2.

The following formula is used to determine the counter value of the sine wave pointer update.

$$MAX\_COUNT = \frac{f_{samp}}{buffer\_size} * f_{\sin}$$

$$= \frac{48000}{24000} * 1 = 2$$

This indicates that the pointer to sine wave lookup table is updated every second sample receive interrupt.

```
f = getCurSineVal()*(maxFlangeSampleDly -1);
```

```
curSinDlyCnt = floorf(f);

f = f - curSinDlyCnt;

incSinPtr();

interpOut = performInterpolation();

output = (0.7*input) + (0.7*interpOut);
```

5.4 Challenges Encountered

5.4.1 Absence of Asynchronous Serial Ports

The ADSP-21364 EZ-KIT development board does not contain any asynchronous serial ports. This posed a great challenge, as two asynchronous serial ports are required in order for the DSP to receive MIDI data. To overcome this obstacle, a UART was implemented using software. The theory and code to perform this function were provided by Analog Devices [13], however they were directed at an older generation of processor. The code was modified to port to the newest generation of SHARC processor. The major section of modification was renaming of registers and recalculation of clock divisors.

"Using the synchronous serial ports (SPORTS) on the SHARC® DSP, it is possible to implement a full-duplex, asynchronous serial port to communicate with UARTs." [13]

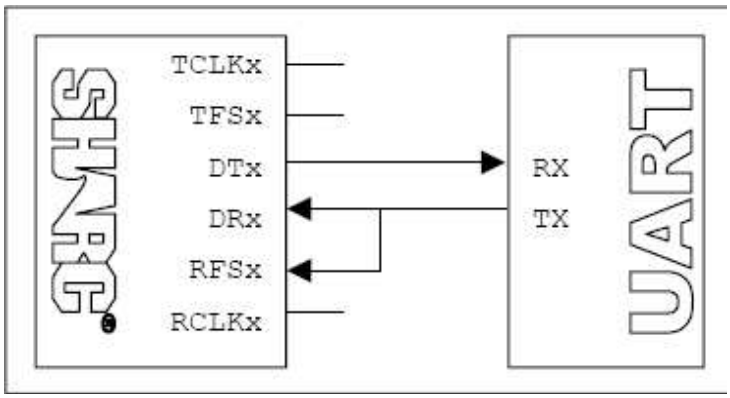

Figure 20. Physical connection between UART and DSP Serial Port [13]

Transmitting data proves to be a simple task. However the processing of receiving asynchronous data synchronously proves to be more complex. In the absence of a frame sync signal, to determine where a new word begins, the transmit data pin of the UART is not only tied to the receive data pin, but also to the frame sync pin. The start bit of the serial data now acts as a frame sync signal.

"As the SPORT has no way of guaranteeing any phase synchronization with the incoming bit stream, it is necessary to over-sample the incoming asynchronous data stream. To accomplish this, the receive clock on the SPORT must be set to three times the desired bit rate." [13]

For communication with a UART device at 31,250 bits/second, the receive clock on the SPORT needs to be set at 3 * 31250, or 93,750 bits/second.

This is achieved by programming the clock divider register *DIVx*, where *x* is the index of the serial port.

$$DIVx = \frac{f_{core}}{4 * f_{baud}} - 1 \quad [13]$$

The core frequency of the ADSP-21364 is 331.776MHz. The desired baud rate is three times the MIDI standard baud, to effectively over-sample.

$$DIVx = \frac{331.776 * 10^6}{4 * (3 * 31250)} - 1 = 884$$

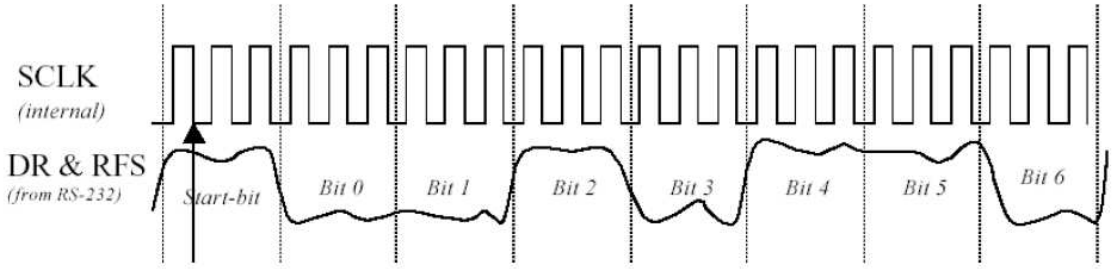

Figure 21. Over-sampling of incoming asynchronous data.

"As the data is over-sampled by a factor of 3, the serial port must be programmed to receive 29 bit words. The first two bits received will be the second and third bits of the frame sync and the remaining 27 bits will represent 9 bits of data (8 bit-word and 1 stop bit) over-sampled by a factor of 3." [13]

The eight data bits are then extracted from this twenty-nine bit word using the *field extract* (*fext*) and *field deposit* (*fdep*) instructions.

5.4.2 Flanger Interpolation

While the algorithm used to reproduce the flanger effect in Matlab produced ideal audio results, when the same algorithm was embedded in the signal processor, sine-based noise was apparent along side the desired output. Through further research and investigation, the cause was identified to be the lack of resolution of the delay used in the effect. In the Matlab algorithm, the delay was calculated in accordance to the sin wave reference. When the calculation yielded a delay that was not an integer number of samples, the result was rounded to the nearest sample. While this produced adequate results on the PC with Matlab, it caused sine-based noise when embedded on the DSP.

To combat this noise the size and resolution of the sine wave lookup table was increased as much as possible. The other tactic to overcome this noise was to introduce interpolation, where a fractional number of samples were required. Linear interpolation was employed at first, which resulted in a vast improvement. To attempt to further improve on this result, four-point cubic interpolation, or Lagrange interpolation was applied to the algorithm.

e.g. If a delay of 94.375 samples is required,

f = 0.375, the fractional component of the delay

x(n) = The value of sample 94.

x(n-1) = The value of sample 93, etc…

$$Interpolated\_Value = \frac{-f(f-1)(f-2)}{6}x(n-1) + \frac{(f+1)(f-1)(f-2)}{6}x(n) - \frac{f(f+1)(f-2)}{6}x(n+1) + \frac{f(f-1)(f+1)}{6}x(n+2)$$

[14]

This formula further increased the quality of the effect output, unfortunately however the background noise could not be removed completely.

5.4.3 Delay Memory

As outlined in *Section 5.3.3*, significant challenges were encountered when trying to implement a substantial delay in internal memory. This was overcome by utilising external RAM. One critical advantage of this approach is that it did not involve lowering the sampling frequency from the professional audio standard of 48 kHz and 24 bits.

5.4.4 Floating Serial Port Clock

Throughout the system development, the signal processing program and the UART and MIDI decode program were developed separately. When executed independently, the programs functioned as expected. However, when combined, the MIDI data was not received correctly.

After extensive investigation the problem was traced to interference within the Signal Routing Unit (SRU). As outlined in *Section 5.4.1*, the UART's receive clock is left floating to implement asynchronous data reception. This floating serial port clock was interfering with data received on the same serial port. Various actions were taken to

reduce the interference produced by the clock pin. Attempts, involving pulling the clock pin high and low, and tying the signal to a separate pin, all failed. Only when the software was bypassed, and the code directly altered to tie the signal to itself was the MIDI data received successfully concurrently with audio processing.

```
SRU(SPORT3_CLK_O, SPORT3_CLK_I);
```

5.4.5 Lack of Documentation

As the ADSP-2136x family of SHARC digital signal processors have only recently been released, the documentation provided is still in the process of re-organisation and as a consequence of this, the learning curve associated with them is very steep.

6. MIDI connections to DSP Board

6.1 Pedal-board to DSP link

The 8051 pedal-board transmits its MIDI messages over a serial link to the DSP effect processor. The 831's standard UART was sufficient to reproduce the MIDI protocol. In default mode eight data bits are sent, LSB first, packed in one start bit (logic '0') and one stop bit (logic '1'). This adheres to the guidelines proposed the MIDI specification [7]. To conform to MIDI protocol the link must also have a baud rate of 31.25 kbps. This poses several challenges. The ADuC831 board's serial port interface set's the baud rate by dividing the core clock. As the board is equipped with an 11.0592 MHz crystal as standard, it is not possible, through any combination of divider coefficients, to achieve the desired baud rate. Therefore the board's crystal was replaced with a 12 MHz oscillator. Operating in Serial Mode 1 (8-bit UART, Variable Baud Rate), timer1 is configured in auto-reload mode, and the timer 1 interrupt is disabled. Using the following equations, outlined in the ADuC831 datasheet [4], the reload value for timer 1 (TH1) and the bit setting for SMOD (double serial baud rate) are established to achieve the desired baud.

$$Baud = \left(\frac{2^{SMOD}}{32}\right) \times (Timer1\_Overflow\_Rate)$$

$$Baud = \left(\frac{2^{SMOD}}{32}\right) \times \left(\frac{Core\_Clock}{12 \times [256 - TH1]}\right)$$

$$SMOD = 0$$
$$TH1 = 255$$

$$Baud = \left(\frac{2^0}{32}\right) \times \left(\frac{12{,}000{,}000}{12 \times [256 - 255]}\right) = \frac{12{,}000{,}000}{32 \times 12} = 31250$$

To ensure UART transmission was functioning correctly, the baud was set to a serial port standard rate of 9600 bps, and methods were inserted to convert the MIDI messages to ASCII values before transmission. The board's UART was then connected to a PC serial port, and the transmissions were observed on a hyper-terminal program, where correct operation was verified.

While the ADuC831's UART functions using logic levels of 5V and 0V, the DSP board's inputs are not 5V tolerant. To ensure correct operation, and to avoid damage to the DSP board, considerable effort was given to the task of logic level conversion from 5V to 3.3V. Several alternatives were considered and attempted, including resistor and zener diode combinations and MIDI optoisolation circuitry. However the best results were produced by utilising a 74LVC373 integrated circuit. 74LVC373 is an "Octal Transparent D-type Latch with 5V Tolerant Inputs". It is essentially eight d-type latches, with 3.3V outputs, with the ability to tolerate 5V logic inputs, making it the ideal translator device.

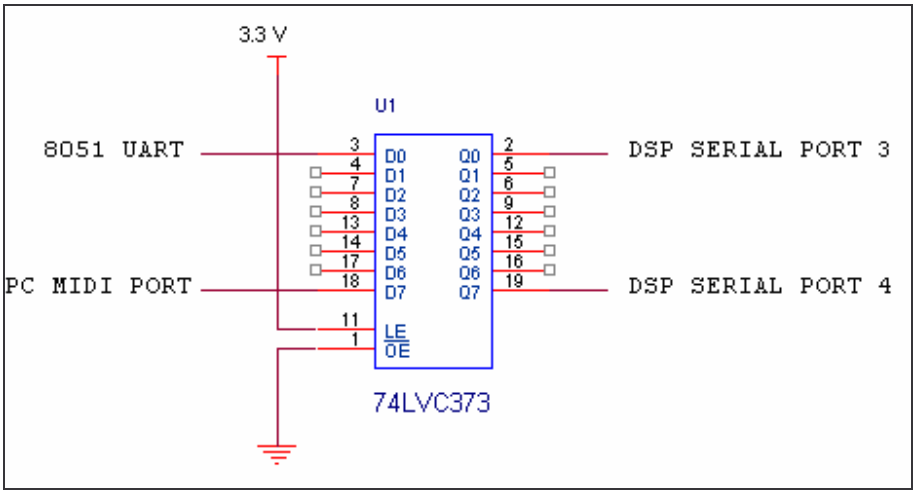

Figure 22. Logic level conversion circuit.

6.2 PC to DSP MIDI Connection

To enable superior effect and patch configuration, the PC GUI sends MIDI configuration data to the DSP board. This is realised through a PC MIDI port. This link is not intended to provide real-time enhancement to guitar playing. Its sole purpose is to configure the device before use. A MIDI to USB converter was used for this project, but a standard sound card MIDI port can perform the same functionality. The MIDI transmission format is a current loop, where 5 mA represents logic '0' and 0 mA represents logic '1'. To convert this to voltage based logic, a standard MIDI optoisolation circuit was assembled.

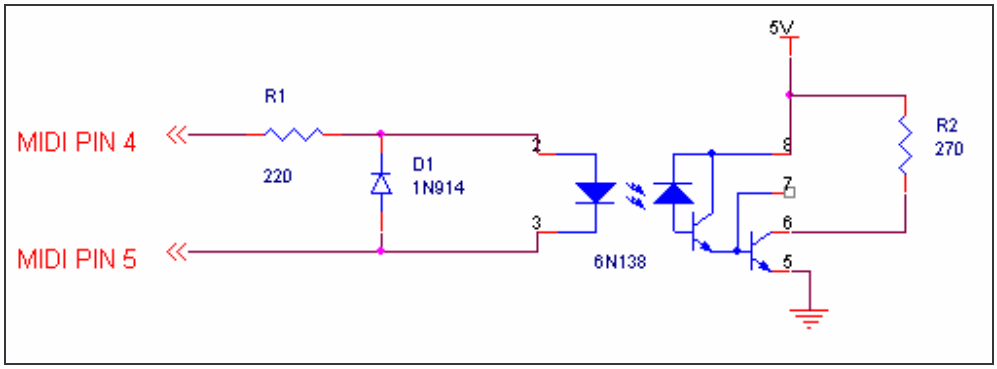

Figure 23. MIDI Optoisolation Circuit

As this circuit outputs 5V logic, a further latch on the 74LVC373 chip was used to convert to 3.3V logic, similar to the 8051 UART conversion.

7. Java Graphical User Interface

A graphical user interface program was coded in Java. Users make alterations to the effect setup, and these changes are sent over the serial link to the DSP. Java was chosen for ease of use, flexibility and provision of excellent API libraries. This program uses the *javax.swing* API to provide a user-friendly interface to effect control, ensuring the underlying MIDI protocol is transparent to the user. The *javax.sound.midi* package of the Java API provides the methods and classes for dealing with MIDI device selection, opening ports, establishing connections and message generation. The java code defines how the setup configuration data is serialised into MIDI messages and transmitted.

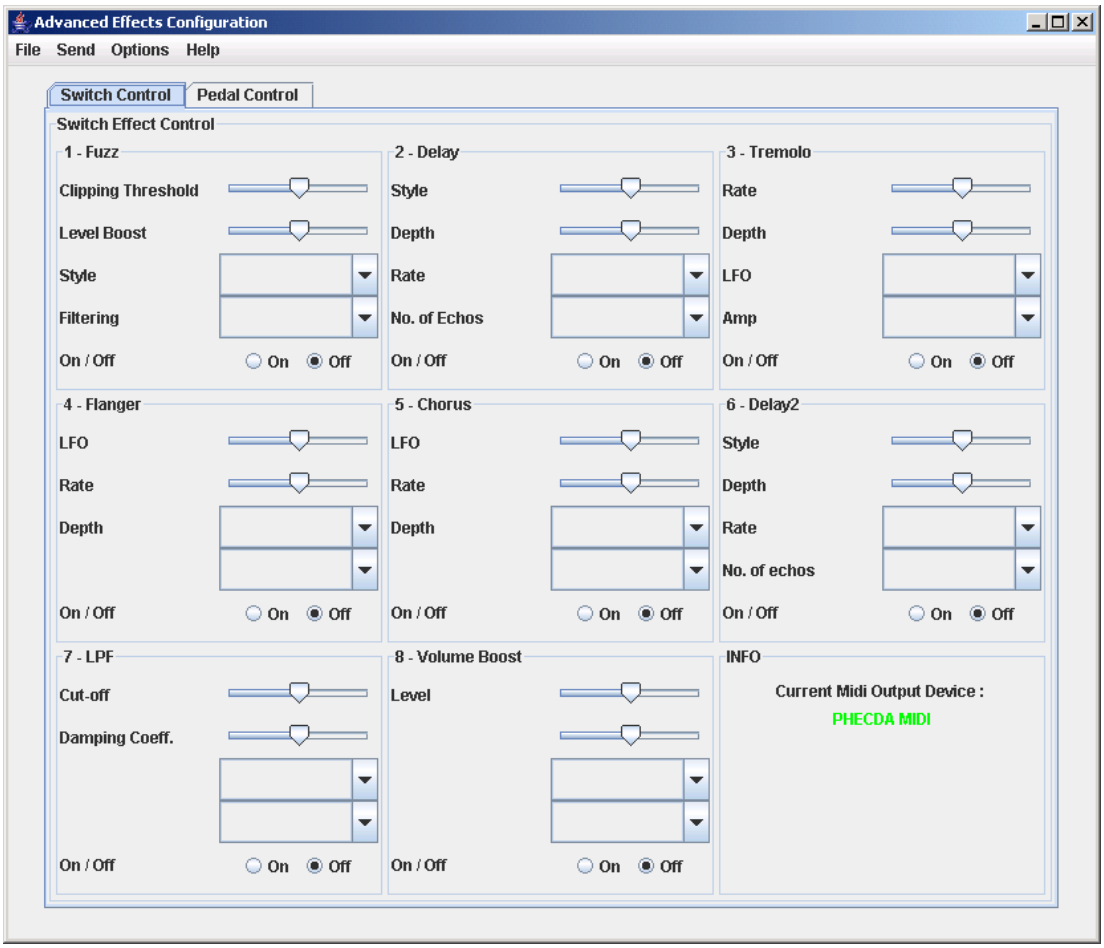

Figure 24. Screenshot of Java Effects Configuration GUI

8. Possibility for Extension

8.1 Wireless Link

It is desirable to have a wireless link between the pedal board and the DSP processor. This would allow a certain amount of freedom when playing, and positioning equipment. The wired serial link has been extensively tested, and RF link would be a beneficial extension to the project. The biggest challenge to establishing a wireless link would be adherence to the MIDI baud rate. Failure to exactly match the 31.25 kbps baud rate means buffering would have to take place. Two *Radiometrix Radio Packet Controllers* would be sufficient to perform this function. While they would not be able to match the exact baud rate, they are designed to be low power and reliable.

8.2 Further research – advanced hand control of effects

Preliminary research was carried out in the area of advanced control of guitar effects using some kind of hand controlled device. This is an area that has much potential. Convention clearly favours foot controlled equipment, however this approach is severely limited, more advanced control could theoretically lead to whole new ways of playing the electric guitar.

Initial brainstorming produced the idea of some format of touch sensor mounted on the face of the guitar. This would be controlled by the strumming or picking hand (right hand for right handed players). One way to approach this idea would be to manufacture the guitar with this sensor built in. This would ease the problems of data transfer and power connections, however it is fundamentally flawed as it cannot be incorporated into existing guitars. An alternative approach could appeal to a much broader market. If the touch-sensor was a stand-alone device, with no power supply and

no wires for data transfer, any guitar owner could install it on the face of their guitar with zero overhead.

For this approach to be feasible, it could make use of RFID (*Radio Frequency Identification*) technology. RFID integrates circuits contain no power supply, a reading devices bounces a signal off it a received its embedded data back.

If this idea could be altered, the pedal board could act as the reading device and poll the touch sensor for its current position. This would mean no replacing of batteries, routing of wires or bulky on board transceiver. It would also consist of a very simple installation procedure. If, after extensive research, this approach is deemed viable, it could be applied to many other areas of remote control. One patent already exists, which encompasses every possible means of guitar control, while RFID is mentioned in a list of possibilities, it does not go into specific detail. [15]

9. Conclusion

This system provides a unique and dynamic environment for guitarists, both novice and professional. A standalone pedal-board offers real-time switching, and parameter change. A PC based GUI offers intricate customisation “off the shelf”, without the need to trawl through oceans of manuals. The adherence to MIDI protocol throughout allows the addition of other standard equipment such as synthesisers and program changers.

This project incorporated a vast range of software, hardware, digital signal processing and embedded systems. Four individual programming languages, and communication between multiple systems made this an exciting and challenging venture.

Table of References

[1] "Using The Low-Cost, High Performance ADSP-21065L Digital Signal Processor For Digital Audio Applications", Revision 1.0 - 12/4/98, Analog Devices, http://www.analog.com/UploadedFiles/Application_Notes/7056820721065L_Audio_Tutorial.pdf

[2] Udo Zölzer, *DAFX Digital Audio Effects*, Chichester, UK, Wiley, 2002.

[3] *Guitar Effects – What they do*, GM Arts, http://users.sa.chariot.net.au/~gmarts/index.htm

[4] ADuC831 data sheet, Analog Devices http://www.analog.com/UploadedFiles/Data_Sheets/90530111ADuC831_0.pdf

[5] 8051 MIDI Pedal-board Controller, 3rd Year Project Report, John O'Dwyer and Norah O'Brian, NUI Galway, 2005

[6] R.A. Penfold, *Advanced MIDI User's Guide,* 2nd Edition, Kent, UK, PC Publishing

[7] Summary of MIDI messages, http://www.midi.org/about-midi/table1.shtml

[8] ADSP-21364 SHARC® Processor Preliminary Data Sheet, Analog Devices www.analog.com/UploadedFiles/Data_Sheets/110075906ADSP_21364_prd.pdf http://www.analog.com/UploadedFiles/Data_Sheets/71670759ADSP_21364_0.pdf

[9] ADSP-21364 EZ-KIT Lite Evaluation System Manual, Analog Devices http://www.analog.com/UploadedFiles/ Associated_Docs/50400613ADSP_21364_EZ_KIT_Lite_Manual_Rev_2.0.pdf

[10] ADSP-2136x SHARC® Processor Hardware Reference, Revision 1.0, October 2005. http://www.analog.com/UploadedFiles/Associated_Docs/ 473915909154761940771310365_HRM_web.zip

[11] Visual DSP++4.0 C/C++ Compiler and Library Manual for SHARC® Processors, Revision 5.0, January 2005, Analog Devices, http://www.analog.com/UploadedFiles/Associated_Docs/16062586440_cc21k_man_whole.zip

[12] "Using External Memory with Third Generation SHARC® Processors and the Parallel Port", *Brian M., Divya S. and Matt W., Rev 2 – March 8, 2005.* http://www.analog.com/UploadedFiles/Application_Notes/141482109EE220v02.pdf

[13] EE-191, Implementing a Glueless UART Using The SHARC® DSP SPORTs, Dan Ledger, Analog Devices, May 6, 2003. http://www.analog.com/UploadedFiles/Application_Notes/399447663EE191.pdf

[14] http://crca.ucsd.edu/~msp/techniques/latest/book-html/node27.html

[15] Guitar effects control system, method and devices*, Harrison Shelton E,* http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US2003196542&F=0

Appendices

Appendix I - Table of Acronyms

ADC	Analog to Digital Converter
ADSP	Analog Devices Signal Processor
ADuC	Analog Devices Micro-Controller
API	Application Programming Interface
bps	Bits per second
DAFX	Digital Audio Effects [2]
DSP	Digital Signal Processor/Processing
GFLOP	Giga-FLoating-point Operations
GUI	Graphical User Interface
ISR	Interrupt Service Routine
I^2S	Inter-IC-Sound
I^2C	Inter-Integrated-Circuit
JTAG	Joint Test Action Group
kbps	Kilo-Bits per second
LFO	Low Frequency Oscillator
LSB	Least Significant Bit(s)
MIDI	Musical Instruments Digital Interface
MSB	Most Significant Bit(s)
PC	Personal Computer
RCA	Radio Corporation of America

RFID	Radio Frequency Identification
RPC	Radio Packet Controller
RTOS	Real-Time Operating System
SHARC	Super Harvard ARchitecture Computer
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

Appendix II – Contents of Project CD

Directory Listing:

Matlab

All Matlab files created to simulate guitar effects.

Sound Samples

All wav output files from Matlab Effect Simulation.

Pedal-board

Assembly code files of the pedal-board operating system.

Embedded System

C and Assembly code of DSP embedded system. All code developed in Analog Devices Visual DSP++ 4.0.

Java

Code associated with the development of the Graphical User Interface.

For a more detailed description, please see *readme.txt* on the project CD.

Appendix III - Project Website

An extensive project website is available for viewing at

http://ohm.nuigalway.ie/02omalley/fyp_0506/

The website consists of 13 pages of project information, updates and links to all related documents and code.