PC based real-time signal processing

Lars Arknæs, TC Electronic A/S
larsa@tcelectronic.com

ABSTRACT

Audio algorithm development typically implies an iteration phase where one or more listening experts adjust and tune the algorithm until the desired performance and quality is achieved. In this phase it is vital that the adjustment done by the expert can be done real-time. A cost-effective fast-prototyping system is proposed, comprising of a PC running MatLab as controller and a dedicated DSP windows DLL to handle real time processing. This solution proved useful for several development applications.

1. INTRODUCTION

Traditionally most real-time signal processing takes place in dedicated hardware, often consisting of A/D converters, D/A converters and DSP's or expensive workstations. And since each algorithm requires its own unique user interface, the control path from user to signal processing often gets complex and inflexible. In many cases a PC is used as prototype interface, using screen, keyboard and mouse.

2. THE PC AS DSP PLATFORM

During the last decade the processing power in a standard office PC has exploded. The core clock frequency in Pentium processors today is 50 times higher than the most common PC processors 10 years ago, so even though the Pentium was not designed to be a DSP it has a lot of processing power, which easily can compete with many dedicated DSP's. Furthermore the signal processing can be performed on the same platform as the user interface, which is very convenient when the listening/tuning process has to take place at external locations (cinemas, theatres etc.). Typically the mapping software between user parameters and signal processing coefficients are written in C++ or another high-level language, but in the algorithm development phase it is convenient to use MatLab as a fast prototyping tool. MatLab includes many built-in signal processing design and analysis functions, which is being used worldwide. Furthermore MatLab can be used to create graphical user interfaces.

3. OBJECT ORIENTED SIGNAL PROCESSING

In audio signal processing many building blocks are used again and again from one algorithm to the next, so an object oriented "LEGO kit" of these building blocks is desired. Such a system has been developed, comprising of many well known signal processing operations such as IIR filter, FIR filter, Fast convolution, Matrix multiplication, soft clipper, quantizer, signal generator, wave recorder, wave player and so on. The system, called *WaveProc*, has been made as a windows DLL which can be called as a function from MatLab.

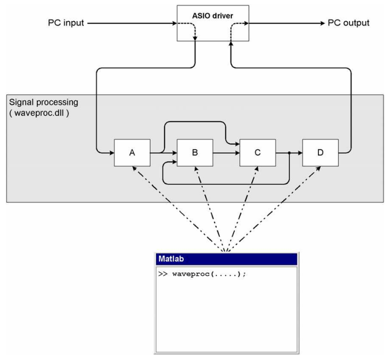

Figure 1 PC based real-time signal processing using MatLab as controller

The signal interface to the PC is an ASIO (Audio Stream Input Output) compatible soundcard, which supports high quality (24 bit) multi channel digital I/O with sample rates up to 96 KHz, which is sufficient for most audio applications. The signal delivered from the ASIO driver is forwarded to the signal processing in the waveproc.dll. The signal processing consists of a desired set of building blocks, picked from the "LEGO kit". Each block is added to the algorithm via the WaveProc function. The number of channels in each block and signal routing between blocks is programmable. Another important issue to address is whether the signal processing in each block should be block based or sample-by-sample based. The block-based choice has the advantage of efficiency, but a major drawback due to the impossibility of making feedback loop in the signal path. That is the main reason why the sample-by-sample solution was chosen. When the feedback loop possibility is present, the MatLab programmer also must decide in which sequence the signal process in each block is executed. In many algorithms, some part must run at a lower or higher rate than the input signal sample rate. This feature has also been included in the WaveProc system, enabling each block to execute either once per input sample, P times per input sample or once every Q input sample.

4. PERFORMANCE

The DSP performance on a PC platform depends on a lot of factors, not only the core clock frequency. In many cases the bottleneck is memory access, particularly when the algorithm needs large blocks of memory, which exceeds the CPU cache, like FIR filters or FFT's. A foundation stone in digital signal processing is the IIR filter. In WaveProc the IIR building block is implemented as a multi-channel cascade of second order sections (biquads), where each channel has the transfer function:

$$H(z) = \prod_{k=1}^{M} \frac{b_{0k} + b_{1k} \cdot z^{-1} + b_{2k} \cdot z^{-2}}{1 + a_{1k} \cdot z^{-1} + a_{2k} \cdot z^{-2}}$$

Both data and coefficients in the filter is 64 bit floating point. On a standard office PC (1.8 GHz Pentium) a cascade of 500 biquads running at a sample rate at 48 KHz in a single channel, consumes 65% of the CPU usage. On a dedicated DSP, a Motorola 24 bit fixed point, a biquad (single precession with first order noise feedback) can be executed on 7 cycles. So when the sample rate equals 48000 KHz and the DSP clock frequency is e.g. 150 MHz, only $(150M/48K)/7 \approx 446$ biquads can be executed per sample. (Of course several hundreds of 24 bit biquads in cascade is not a very wise strategy, due to the noise level).

The MatLab control code used for the implementation of the biquad cascade is:

```
%--------- ASIO driver setup --------------------
waveproc('setdriver','ASIO DIGI9636/52');
waveproc('open',48000,25,25);
%--------- Create objects based on CElement ------
inBlk  = waveproc('add_element', 'source')
iirBlk = waveproc('add_element', 'iir_filter')
outBlk = waveproc('add_element', 'destination')
%--------- Set execution sequence ----------------
waveproc('set_exeseq', [inBlk iirBlk outBlk -1]);
%--------- Set number of outputs -----------------
waveproc('set_no_outputs', inBlk,  1);
waveproc('set_no_outputs', iirBlk, 1);
waveproc('set_no_outputs', outBlk, 1);
%--------- Set routing (sources) -----------------
waveproc('set_source', inBlk, 1, 1);
waveproc('set_source', iirBlk, inBlk, 0);
waveproc('set_source', outBlk, iirBlk, 0);
%--------- Set algorithm specific properties -----
NBIQ = 500;
waveproc('set_data',iirBlk,'no_biquads',NBIQ);
coef = ([1 0 0 1 0 0]'*ones(1,NBIQ))';
waveproc('set_data',iirBlk,'coefficient',coef);
%--------- Let the show begin --------------------
waveproc('start');
```

5. CONCLUSION

Compared to other real-time signal processing tools like *MatLab/Simulink*, *WaveWarp*, *SoundWeb* or *Reactor*, the WaveProc system enables you to implement your own (sample-by-sample) signal processing in a relatively efficient language (C++) and utilizes the flexibility and ease-of-use of MatLab as a controller. A major drawback of the system (when running under windows) is the latency from digital input to digital output, which makes the system inapplicable if e.g. the system is located in a feedback loop. This latency is due to input/output buffers, which protects the signal processing against sudden task shifts of the operating system. The need for large protection buffers (hence latency) could perhaps be reduced if the system was moved to run under a more predictable operating system, like e.g. Linux.