RISC OS Digital Signal Processor

Introduction

RDSP provides a powerful sound engine for RISC OS with both easy to use features as well as advanced capabilities.

Small simple BASIC programs are included to illustrate features. For example:

- Sample load and playback
- Effects algorithms
- A 16 channel digital synthesiser feature multiple waveforms, filters and modulation
- BBC Micro sound emulation

Sound samples can now be easily loaded and played back from a few built in commands.

The ENVELOPE command can be used to create sound effects using a unique synthesiser.

Features

- Sample playback
- Sound effects : Reverb, Chorus, Echo
- Synthesis :
    - Saw, pulse, triangle and pulse-noise
    - Waveform modulation
    - Waveform mixing
    - Sample modulation
    - Ring modulation

    - Low pass resonant filtering
- Recording RDSP;s output to a file for easy playback and exporting to other platforms.

BASIC Operation

The SOUND command

LEGACY BBC MICRO FEATURES

The SOUND command may be invoked with a BBC Micro style 4 bit volume on any of the available channels (0 to 15). In this case the SOUND command will default to producing a square wave on any channel other than channel zero. Channel zero will produce noise or a narrow pulse i.e. a click sound e.g.

SOUND 1,-15,172,50

Plays middle C for half a second.

This is equivalent to SOUND 1,-15,101,10 on the BBC Micro.

SOUND 0,-15,0,50

Plays a narrow pulse wave for half a second

SOUND 0,-15,4,50

Plays noise for half a second.

n.b. The noise on channel 0 may be modulated by channel 1, provided that channel 0 is re-triggered e,g,

SOUND 1,0,0,50 : SOUND 0,-8,7,50

Differences Between Legacy Mode and a BBC Micro

RDSP implements 16 channels rather than 4 and 365 1/8ths of a tone are supported rather than 255 Hence middle C requires a higher number than a BBC micro..

RISC OS SPECIFIC ENHANCEMENTS

RDSP is a virtual sound chip that supports:

- 4 waveforms :

    - Saw,
    - A pulse variable wave between 1% and 50% duty cycle
    - Triangle
    - Pulse noise
- PCM sample playback
- Low pass filtering
- XOR of channels to produce complex interference effects.
- AND of waveforms to produce complex tones
- 128 PCM sample envelopes
- 128 waveform envelopes
- Stereo sound placement
- Chorus, delay and reverb effects
- Recording to disc.

The SOUND command has the following fields in RDSP. Where the letters below represent hex digits:

SOUND PFIC, TTVV, FRNN, DDDD

P = Pan 0 = centre. Pan &1000 = max left. Pan &f000 = max right
F = Flags : &100 - Synchronise,
&200 - Mute,
&400 - XOR with previous channel
&800 - Queue sound (up to 365 sounds may be queued per channel)
I = Instrument :
&00 - Waveform,
&10 - Envelope,
&20 - Sample
C = Channel : &0 to &f

T = Tone : Waveform mask or Envelope number or Sample number

n.b. a waveform of 0xff is disallowed in order to support
legacy OS Word 7 behaviour.

If I = &00 then The following waveforms may be selected via a bitmask:

RDSP will AND the resulting waveforms together:

&01 to &1f : Pulse wave from ~1% to a square wave

i.e. &1f is a square wave like the BBC Micro

e.g. `SOUND 1,-15,100,20` is the same as

`SOUND 1,&1fff,100,200`

&20 : Saw wave i.e. a 'ramp'.

&40 : Triangle wave

&80 : Modulatable noise

Where I are values in the most significant byte i.e. &0000 to &ff00

If I = &10 then T will select ENVELOPEs from &00 to &ff

If I = &20 Then T will select samples from 0 to &ff

V = Volume : &00 to &ff

F = Low pass filter cut off : 0xf000 to 0x0000 for 16 values

R = Resonance (except for the least significant bit which is used to extend N)

i.e. R = &0e00 i.e. valid values are 0x0200,0x0400, 0x0600, 0x0800, 0x0a00, 0x0c00, 0x0e00

N = Note - Range 0 to 364 i.e. &000 to &16c

D = Duration of sound in 100ths of a second.

The ENVELOPE command

The ENVELOPE command has different parameters for RDSP:

ENVELOPE N,W,F,O,T,S2,S13,D1,D2,D3,A,D,S,R

N = envelope number:

* 0 to 127 for envelopes based on waveforms.
* 128 to 255 for envelopes based on samples.

W = Same as sound command however 255 is allowed.

F = Most significant 4 bits specify filter cut off

where 0xf0 is fully open and 0x00 is closed

Least signficant 4 bits specify Q

O = envelope modulation options :

0x01 - ADSR pitch modulation

0x02 - ADSR filter modulation

0x04 - ADSR PWM modulation

0x08 - 3 stage envelope amplitude modulation

0x10 - 3 stage envelope pitch modulation

0x20 - 3 stage envelope filter modulation

0x40 - 3 stage envelope pulse width modulation

0x80 - Loop 3 stage envelope

T = time of 3 stage envelope in 100ths of a second

S2 = number of steps in stage 2 of 3 stage envelope

S13 = number of steps in stage 1 and 3 of 3 stage envelope

D1 = modulation of parameters selected by O for 3 stage envelope in stage 1

D2 = modulation of parameters selected by O for 3 stage envelope in stage 2

D3 = modulation of parameters selected by O for 3 stage envelope in stage 3
A = ADSR attack amount. Larger values are longer
D = Decay amount. Larger values relative to S create a pluck effect.
S = Sustain volume proportion. The duration is set by the sound command 'D'
Larger values are louder and sustain volume is proportional to 'V'
in the sound command.
R = Release time. Larger values are longer.

STAR COMMAND REFERENCE

***Rstart**
Start RDSP engine

***rstop**

Stop RDSP engine

***rfilter** <channel> <cut-off> <resonance>

Configure Low Pass Filter for channel

***rsound** <channel> <flags> <wave | instrument> <volume> <pitch> <duration>

***renvelope** <number> <name> <wave> <cutoff-resonance> <options/ASR-steptime> <ASR steps> <AR delta> <ADSR levels>

Define instrument to play via sound command. Al values are in hex e.g. for ADSR 20808020 is attack=&20, decay=&80, sustain=&80 and release=&20

***rfxmix** <channel | 255 (all off)> <0|1>

Mix sound channel into the effects processor input.\nSyntax\tRFXMix

***rfx** <algorithm mask>

Turn on sound effects algorithms

4 = off 8 = Delay. 16 = Chorus, 32 = Reverb

***rdelay** <time in centiseconds> <feedback>

Delay (echo) effect configuration

***rload** <sample bank> <pathname>

Load a 44khz 16 bit sample into sample memory bank

***rlist**

List envelopes in memory including all settings defined internally as per the *renvelope command.

***rvolume** <level>
Master volume control (0-256)

***rlisten** <time>
Records RDSP output to a memory buffer. Length of time is in centiseconds of a second (0-120000).
NOTE: If the module area does not have sufficient memory and a high value is chosen RISC OS may hang at present.

***rnolisten**
Cancels RDSP output recording

***rsave** <pathname>
Save in-memory recording of RDSP to disc.

***rsampleloop** <sample id> <loop-point> <loop-to>
Turn on sample looping:\nSyntax:\trsampleloop <sample id> <loop-point> <loop-to>

***rbell** <sample-id>
Override system bell with sample. Please note that this will affect any sound played at the frequency and channel of the bell.

***rpan** <channel> <pan>
Pan sound from left (0) to right (255). Centre = 128 for channel 1 to 16.

***rchorus** <rate 1-16> <depth 0-255>
Chorus effect configuration

***rhall** <size 0-255>
Hall effect reverb algorithm configuration

***rroom** <size 0-255>
Room effect reverb algorithm configuration:

TECHNICAL SPECIFICATION

RDSP uses the shared sound feature within RISC OS and runs at 44Khz using 16 bit audio. All DSP algorithms are implemented using fixed point maths with no floating point used.

Internally the `buffer_fill()` function within c.DSP implements the synthesiser and the state of the synthesiser is stored within a global variable referenced as "`extern RISC_SYNTH g_synth`". This global variable is configured by the module interface c.cModule which handles OS_WORD 7 and 8, implements the SWIs and the star commands.

For speed and simplicity, for `buffer_fill()`, only the default 100 bytes of stack are used and the code is a flat function with no decomposition into sub-functions employed.

The synthesiser implements 16 sound channels each of which consist of :

- A phase accumulator oscillator which transform the accumulator value into AND'd combinations of::
    - Saw
    - Triangle
    - Pulse
    - Pulse noise implemented using a trivial pseudo-random number generator based on linear congruential generator.
- Alternatively pulse code modulation(PCM – WAV file) sample playback may be performed at either the original speed the sample was recorded or an alternate frequency i.e. samples may be pitched down or pitched up.
    - Samples may also be looped and thus sample & synthesis complex waveforms can be employed or long tails on a piano sound - where the end of a piano key hit is looped and is then faded away by an envelope.
- A four stage attack-decay-sustain-release (ADSR) envelope that uses a simple linear counter for each state and a simple state counter. This is locked to amplitude modulation but can also optionally be used for pitch, filter and pulse modulation.
- A three stage envelope that also employs a simple linear counter intended for pitch, amplitude, filter and pulse width modulation. This envelope may be optionally looped to implement complex modulation effects.or it can be used emulate a low-frequency oscillator.
- A trivial infinite impulse response 2 pole resonant filter is implemented using fixed point maths.

There is a single effects bus which chains dry/wet audio into each of three effects in a fixed order: CHORUS -> DELAY -> REVERB.

These algorithms use simple ring buffers which implement delay lines to operate.

Chorus employs a delay line that records the input signal and modules its record/playback rate between normal speed and a reduced speed. Hence the waveform played back is both pitched up and down against the dry signal. This is because a sound recorded at a slower sample rate which is then played at normal sample rate will have its pitch raised. Similarly, a sound recorded at a normal sample rate which is then played at a slower sample rate will have its pitch lowered. Hence the comb filtering/detuning of a chorus effect is implemented.

Delay is formed of a single delay line implemented by a ring buffer that plays back a recorded sound after a short delay through the ring buffer. Some of the output signal is fed back into the ring buffer allowing an echo to be played with a reduction in amplitude for each repeat.

Reverb is formed of multiple delay lines each of which has a relatively prime delay time. The output of the delay line is attenuated, has its phase inverted, is fed through a filter to ‘colour’ the sound. Finally the output of each delay line is fed into every other delay line creating a complex series of echoes.

KEY DSP CODE FRAGMENTS:

The Low Pass Filter

```
// resup - resonance feedback needs to increase as filter reduces
feedback when cutoff is higher.

res = 12 - lpf.resonance;
signali = lpf.resonance == 0 ? (signal << 4) : (signal << 4) -
           ((lpf.pole2*resUp) >> res); // res is logarithmic

// This code describes fractional changes in signal
// without using divide instructions.
lpf.pole1 = (lpf.pole1 * (256 - lpf.cutoff) >> 8) +
            ((signali * lpf.cutoff) >> 8);
lpf.pole2 = (lpf.pole2 * (256 - lpf.cutoff) >> 8) +
            ((lpf.pole1 * lpf.cutoff) >> 8);


int rout = lpf.pole2 >> 4;
```

The Oscillator

```
// pulse / square
if (wave & 0x1f)
{
  signal = osc_control < pulse_width ? SWITCH_TOP_AMP :
```

```
                                        SWITCH_BASE_AMP;
}
// saw wave
if (wave & WAVEFORM_SAW)
{
  // AND waveforms together
  if ((wave & 0xdf)&&signal!=0)
    signal &= osc_control - HALF_SAMPLE_RATE;
  else
    // or just assign this waveform
    signal = osc_control - HALF_SAMPLE_RATE;
}

// triangle wave
if (wave & WAVEFORM_TRIANGLE)
{
  if ((wave & 0xbf)&&signal!=0)
  {
    int s = osc_control < HALF_SAMPLE_RATE ?
                                osc_control :
                                SAMPLE_RATE - osc_control;
    s -= QUARTER_SAMPLE_RATE;
    signal &= s;
  }
  else
  {
    signal = osc_control < HALF_SAMPLE_RATE ?
                                osc_control :
                                SAMPLE_RATE - osc_control;
    signal -= QUARTER_SAMPLE_RATE;
    signal = signal * 2; // triangle is quieter -
                         // normal raise- to +11k.
  }
}
// modulatable noise (logarithmic)
if (wave & WAVEFORM_NOISE)
{
  // allow LCG prng to repeat last value mix into the signal
  // 41860090 == run every time
  //   218267 == lowest frequency
  channel->m_noise_throttle += synth_channel->tonetable[tone];
 if (channel->m_noise_throttle >=
                 synth_channel->tonetable[MAX_TONE-1])
 {
  channel->m_noise_throttle -= synth_channel->tonetable[MAX_TONE-1];
  // Trivial linear congruential generator.
  // Using the sample constants as Borland C for now
  channel->m_prng = (22695477 * channel->m_prng) + 1;
  // prng is a 2^32 number, so as usual Modulus is automatic.
 }
 if ((wave & 0x7f)&&signal!=0)
 {
   unsigned int tmp = channel->m_prng >> 31;

   if (tmp & 1)
   {
```

```
      signal &= SWITCH_BASE_AMP;
    }
    else
    {
      signal &= SWITCH_TOP_AMP;
    }
  }
  else
  {
    unsigned int tmp = channel->m_prng >> 31;

    if (tmp & 1)
    {
      signal = SWITCH_BASE_AMP;
    }
    else
    {
      signal = SWITCH_TOP_AMP;
    }
  }
}
```