Developing Supercollider Unit Generators Under GNU/ Linux - Part 2, Square wave oscillator

Matthew John Yee-King

June 4, 2007

Contents

1 Introduction

In this tutorial, I'll write a basic Square wave UGen based on the code from the MySaw Ugen in part 1.

2 Edit the SConstruct build script

First off, we need to add another item to the SConstruct build script. In the section 'plugs', we add MySquare:

```
plugs = [
'MySaw',
'MySquare'
]
```

3 Edit the source code

Next we edit the MYSaw.cpp source code file so it computes a square wave instead of a saw tooth.

1. Create a new file called MySquare.cpp in the source folder. Copy the contents of 4.1 into it. This is the skeleton for the new UGen.

2. Now how can you compute a square wave? Well the most basic method I could come up with was to generate 0s for wavelength/2 then 1s for wavelength/2. This will result in a DC offset, but that's ok for now. Since the number of samples of output we compute in each call to next_a will not match the wavelength, the unit generator will need to remember how many 0s or 1s it has output and whether it is outputting 0s or 1s at the moment. How to remember the state of the UGen? in the struct extension...

3. Add two class fields to the struct block - waveState and written. They'll need to be initialised in the 'constructor', so add lines to the constructor to set initial values to 0.

4. Ok now to compute the waveform, first with an audio rate frequency input. We need to calculate how many 0s or 1s we should output from the current frequency, i.e. the pulse width p in terms of samples:

$$p = \frac{r}{2f} \tag{1}$$

   where p = pulse width, f = frequency and r = sample rate in samples per second

   The code for this is:

```
pw_s = 1/freq[i] * SAMPLERATE * 0.5;
```

   where freq is a pointer to the input array, pw_s is a local variable and SAMPLERATE is a global server config variable.

5. Now we need some code which generates the output of the UGen using the parts we put in place above:

```
void MySquare_next_a(MySquare *unit, int inNumSamples)
{
  // get the pointer to the ouput buffer
  float *out = OUT(0);
  // get the pointer to the input buffer
  float *freq = IN(0);
  // a local variable to store the current pulse width
  float pw_s;
```

```
  // a local variable to remember if we are outputting 0s or 1s
  int waveState;
  // a local variable to store the number of 1/0s written
  int written;
  // pull state out of the struct into local variables
  // this is quicker than repeatedly referring to the struct.
  written = unit->written;
  waveState = unit->waveState;

  // loop through the input array:
  for (int i=0; i < inNumSamples; ++i)
    {
      // calculate instantaneous pulse width
      pw_s = 1/freq[i] * SAMPLERATE * 0.5;
      // have we written enough 1s or 0s?
      if (written > pw_s){
        // yes, flip state
        waveState = 1-waveState;
        written = 0;
      }
      else {
written++;
      }
      // write a sample
      out[i] = waveState;
    }
  // store the state back out to the struct for
  // next time.
  unit->waveState = waveState;
  unit->written = written;
}
```

Things to note here

(a) You don't write to the output till you have finished with the input as an optimisation on the server means the input might be the same array as the output!

(b) It is more efficient to store UGen state as local variables which are read once from the struct at the start and written back at the end as opposed to repeatedly reading and writing to the struct.

6. Now you need to build this new UGen - scons in top level directory.

7. Now define another class on the sclang side - you can adapt your code for the MySaw class. (4.2).

8. That is basically it! Check the full code (4.3) for the function which deals with the control rate input. It is really the same except it only computes the pulse width once per block as it only changes once per block.

9. **Challenge!** Can you write a DC corrected version of the square wave UGen which outputs either -1 or 1 as opposed to 0 or 1 like this one?

4 Code

4.1 MySquarestripped.cpp

```
#include "SC_PlugIn.h"

static InterfaceTable *ft;

struct MySquare : public Unit
{

};

extern "C"
{
  void load(InterfaceTable *inTable);
  void MySquare_next_a(MySquare *unit, int inNumSamples);
  void MySquare_next_k(MySquare *unit, int inNumSamples);
  void MySquare_Ctor(MySquare* unit);
};

void MySquare_Ctor(MySquare* unit)
{
  // 1. set the calculation function.
  if (INRATE(0) == calc_FullRate) {
    // if the frequency argument is audio rate
    SETCALC(MySquare_next_a);
  } else {
    // if the frequency argument is control rate (or a scalar).
    SETCALC(MySquare_next_k);
  }

}


void MySquare_next_a(MySquare *unit, int inNumSamples)
{

}
```

```
void MySquare_next_k(MySquare *unit, int inNumSamples)
{

}

void load(InterfaceTable *inTable)
{
  ft = inTable;
  DefineSimpleUnit(MySquare);
}
```

4.2 MySquare.sc

```
MySquare : UGen {

  *ar { arg freq = 440.0, iphase = 0.0, mul = 1.0, add = 0.0;
^this.multiNew('audio', freq, iphase).madd(mul, add)
  }

  *kr { arg freq = 440.0, iphase = 0.0, mul = 1.0, add = 0.0;
^this.multiNew('control', freq, iphase).madd(mul, add)
  }

}
```

4.3 MySquare.cpp

```
#include "SC_PlugIn.h"
// InterfaceTable contains pointers to functions in the host (server).
static InterfaceTable *ft;

struct MySquare : public Unit
{
//  double pWidth; // pulse width for the squarewave
  int waveState; // on or off?
  int written; // how many on samples have I written?
};
// declare unit generator functions
extern "C"
{
  void load(InterfaceTable *inTable);
  void MySquare_next_a(MySquare *unit, int inNumSamples);
  void MySquare_next_k(MySquare *unit, int inNumSamples);
  void MySquare_Ctor(MySquare* unit);
```

```
};

void MySquare_Ctor(MySquare* unit)
{
  // 1. set the calculation function.
  if (INRATE(0) == calc_FullRate) {
    // if the frequency argument is audio rate
    SETCALC(MySquare_next_a);
  } else {
    // if the frequency argument is control rate (or a scalar).
    SETCALC(MySquare_next_k);
  }
  // start in the off position
  unit->waveState=0;
  // how many on samples have we written?
  unit->written = 0;
  // map freq to onCount
  //  unit->freqMult = SAMPLERATE1/SAMPLEDUR;
  MySquare_next_k(unit, 1);
}

void MySquare_next_a(MySquare *unit, int inNumSamples)
{
  // get the pointer to the output buffer
  float *out = OUT(0);
  // get the pointer to the input buffer
  float *freq = IN(0);
  float pw_s;
  int waveState;
  int written;
  written = unit->written;
  waveState = unit->waveState;

  // pulse width in samples
  //  pw_s = 1/freq[0] * SAMPLERATE * 0.5;
  for (int i=0; i < inNumSamples; ++i)
    {
      pw_s = 1/freq[i] * SAMPLERATE * 0.5;
      // have we written enough 1s or 0s?
      if (written > pw_s){
// yes, flip state
waveState = 1-waveState;
written = 0;
      }
      else {
written++;
```

```
      }
      // write a sample
      out[i] = waveState;
    }
  unit->waveState = waveState;
  unit->written = written;
}

void MySquare_next_k(MySquare *unit, int inNumSamples)
{
  // get the pointer to the output buffer
  float *out = OUT(0);
  // get the pointer to the input buffer
  float *freq = IN(0);
  float pw_s;
  int waveState;
  int written;
  written = unit->written;
  waveState = unit->waveState;
  // at control rate, the freq is read from the first number in the
  // input buffer, as freq only gets written once per buffer length.
  pw_s = 1/freq[0] * SAMPLERATE * 0.5;
  // pulse width in samples
  //  pw_s = 1/freq[0] * SAMPLERATE * 0.5;
  for (int i=0; i < inNumSamples; ++i)
    {
      // have we written enough 1s or 0s?
      if (written > pw_s){
// yes, flip state
waveState = 1-waveState;
written = 0;
      }
      else {
written++;
      }
      // write a sample
      out[i] = waveState;
    }
  unit->waveState = waveState;
  unit->written = written;
}

////////////////////////////////////////////////////////////////////////
// the load function is called by the host when the plug-in is loaded
void load(InterfaceTable *inTable)
{
```

```
  ft = inTable;
  DefineSimpleUnit(MySquare);
}
////////////////////////////////////////////////////////////////////////
```