TMS320 DSP
DESIGNER'S NOTEBOOK

How Can Comb Filters be Used to Synthesize Musical Instruments on a TMS320 DSP?

APPLICATION BRIEF: SPRA252

Leor Brenman
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
February 1995

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Examples

How Can Comb Filters be Used to Synthesize Musical Instruments on a TMS320 DSP?

Abstract

Music synthesis is the ability to create musical scores by synthesizing different musical instruments. Different methods of music synthesis include sampled sound synthesis (wavetable synthesis), FM synthesis, and instrument modeling. Sampled sound synthesis inherently requires significant amounts of memory to store instrument samples but results in extremely natural-sounding music. FM synthesizers are algorithmic and typically require little memory but result in unnatural-sounding music. Instrument models, based on analysis of the instrument being synthesized, often yield efficient implementations producing highly-natural sounds.

This document presents a DSP implementation of a model for synthesizing plucked strings, based on the Karplus Strong Plucked-String Synthesizer.

A system block diagram and several code listings are provided.

Design Problem

Music synthesis is the ability to create musical scores by synthesizing different musical instruments. Different methods of music synthesis include sampled sound synthesis (wavetable synthesis), FM synthesis, and instrument modeling. Sampled sound synthesis inherently requires significant amounts of memory to store instrument samples but results in extremely natural-sounding music. FM synthesizers are algorithmic and typically require little memory but result in unnatural-sounding music. Instrument models, based on analysis of the instrument being synthesized, often yield efficient implementations producing highly-natural sounds.

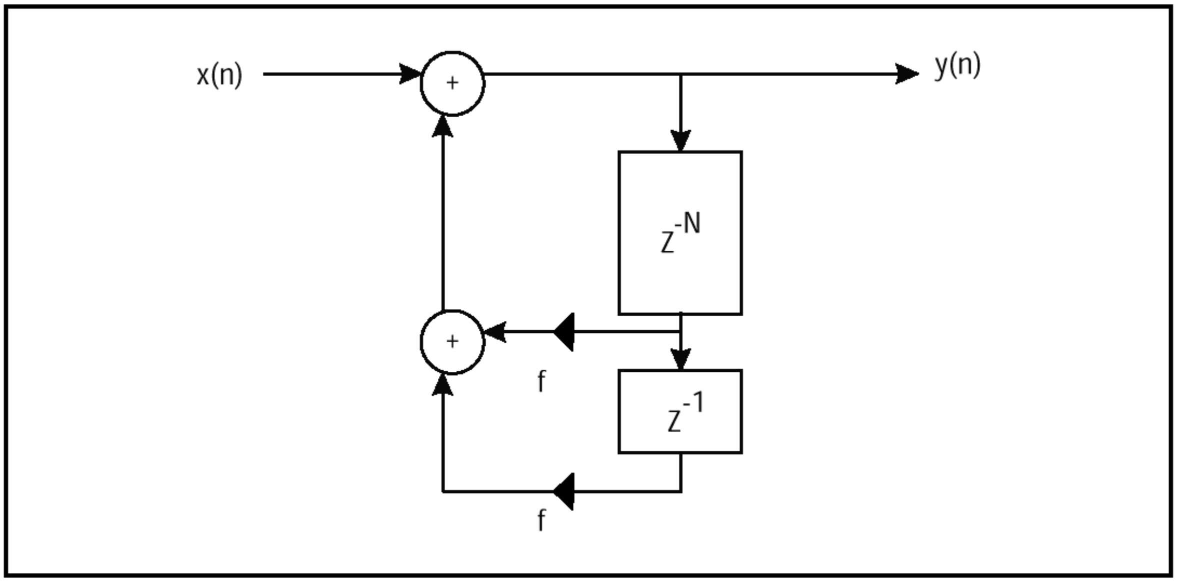
This document presents a DSP implementation of a model for synthesizing plucked strings, based on the Karplus Strong Plucked-String Synthesizer.

Solution

String-Synthesis Model

The Karplus Strong string-synthesizer model produces extremely natural sounding plucked strings. The model is based on a IIR comb filter, shown in Figure 1.

Figure 1. Karplus Strong String-Synthesizer Model



The input, x(n), is a burst of Guassian White Noise lasting N samples and is zero elsewhere. The pitch period of the synthesized plucked string is N times the sample period. The output, y(n), is the synthesized plucked string sound and is valid after the Nth input sample. The multipliers, f, must be less than or equal to 0.5 for IIR stability, and determine the sustain of the synthesized string sound. A value of 0.5 represents the longest sustain. The timbre of the sound is similar to that of a plucked steel-string guitar.

An alternate view of the model is to load the tap delay line with white noise and let the filter ring as long as desired with no input[x(n) = 0]. In this case, the output is valid as soon as the filter is started.

The synthesized string sound has a frequency of fs/N, where fs is the system sampling frequency. This inverse dependence on N results in poor granularity for small N. That is, a unit change in N when N is small results in a large change in the frequency of the synthesized string.

This model, while simple, is suprisingly realistic; the burst of noise represents the plucking of the string and the comb filter, which acts as a resonator, represents the resonating body of an string instrument such as an acoustic guitar or bass. This model is well suited for implementation on DSPs which are designed for implementing digital filters.

DSP Implementation

This type of filter is easily implemented on Texas Instruments DSPs. Many TI DSPs support circular buffering which facilitates implementing the tap-delay line of the comb filter efficiently. Also, they provide the necessary numerical processing speed to support 44.1-kHz sampling-rate processing required for CD-quality audio. On-chip peripheral support for analog-to-digital and digital-to-analog converters reduces chip count and system cost.

For example, the TMS320C31 DSP, with a 33-75 ns instruction cycle time, circular buffer support, and on-chip serial port, is well-suited for implementing these types of algorithms. The following code segment implements one iteration (sample) of the string synthesizer model on the ’C31.

Example 1. Code Listing

```
;AR0 points to tap delay sample y[n-N]
;BK = pitch period value, N+1
;R1 = sustain (0.5=long,0.48=med,0.45=short)
   LDF   *AR0++(1)%,R0    ;Load y[n-N]
                          ;AR0 points to [n-(N-1)]
ADDF     *AR0—(2)%,R0     ;Add y[n-(N-1)]
                          ;AR0 points to y[n]
MPYF     R1,R0            ;f*[y(n-N) + y(n-(N+1))]
STF      R0,*AR0++(2)%    ;Store R0 = y(n)
```

```
                          ;AR0 points to y[n-(N-1)]
                          ;= (next) y[n-N]
* R0 contains the return value, y(n)
```

When a new plucked string is desired, a circular buffer of the appropriate size is setup and filled with white noise. The appropriate parameters, such as the pitch period and sustain factor, are set and remain fixed between calls to the string-synthesizer routine. Samples should be produced at 44.1 kHz for CD-quality music synthesis.

As shown in the preceding code segment, the core of the algorithm can be executed in four ’C31 insruction cycles per output sample. The overhead that is required is for setting up the parameters and loading of the buffer with white noise upon creation of a new string and the interrupt service routine associated with writing the output to the ’C31 serial port for delivery to the D/A converter for listening.

A C-callable version of the string synthesizer function is shown in Figure 3. A C-calling shell is shown in Figure 4.

The TMS320DSP BBS file KPSTRONG.EXE contains the necessary files to create a library of C-callable functions for implementing a string synthesizer using the algorithm described in this paper. It is written primarily in C and C-callable assembly language. A TMS320C30 EVM demo is also included. This code serves as an example of an instrument modeling music synthesis algorithm as well as an example of implementing circular addressing in C and C-callable assembly language.

Example 2. C-callable Version of String Synthesizer Function

```
* Strfunc.asm -  Karplus Strong Base Model Plucked String Synthesizer
*                implementation in TMS320C3x/’C4x assembly language.
*
*                KPSTRING data structure:
*
*typedef struct kpString {
*    DTYPE *tapDelay;       /* Tap delay buffer data pointer */
*    DTYPE sustainFactor;   /* Sustain factor, f 0.5 */
*    int pitchPeriod;       /*Pitch period in samples = fs/fdesired*/
*    DTYPE *tapDelayBase;   /* Tap delay buffer base pointer */
*    int currLoc;           /* Index points to delay[N+1] */
*    DTYPE *noise;         /*Noise buffer used to initialize string*/
*} KPSTRING;
*
*                Function prototype:
*
* float string(KPSTRING *st, float *out, int nsamples);
*
```

```
*
FP .set AR3
                 .global _string
_string
*
* Stack manipulation
*
    PUSH     FP
    LDI      SP,FP
    LDI      *-FP(2),AR2  ;AR2 points to a KPSTRING data structure
    LDI      *-FP(3),AR1  ;AR1 points to output[0]
    LDI      *-FP(4),RC   ;RC = nsamples
*
* Setup for circular buffer fetches and loop
*
    LDI      *+AR2(0),AR0 ;AR0 points to y[n-N]
    LDF      *+AR2(1),R1  ;R1 = decay (0.5=long,0.48=med,0.45=short)
    LDI      *+AR2(2),BK  ;BK = pitch = N = BufferSize-2

    ADDI      2,BK        ;BK = BufferSize
    SUBI      1,RC        ;RC = 1 less than # iterations for RPT
*
* Implement Comb buffer
*
    RPTB      STLOOP         ;loop over n
    LDF       *AR0++(1)%,R0  ;R0 = y(n-N)
                             ;AR0 points to y[n-(N-1)]
    ADDF      *AR0—(2)%,R0   ;R0 = y(n-N) + y[n-(N-1)]
                             ;AR0 points y(n)
    MPYF      R1,R0          ;R0 = f*[y(n-N) + y(n-(N-1))]
    STF       R0,*AR0++(2)%  ;R0 = y(n), store result in delay line
                             ;AR0 points to y[n-(N-1)]
                             ; = (next) y(n-N)

* Uncomment out these two lines if this routine should ADD it’s
*  calculated output to the output stream, otherwise the output is
*  overwritten

*   LDF      *AR1,R2         ;Get output buffer data value
*   ADDF     R2,R0           ;Add calculated value

STLOOP:
    STF      R0,*AR1++(1)    ;Store y(n) to output buffer

*
* Restore circular buffer pointer
*
*** STI AR0,*+AR2(0)         ;restore circular buffer pointer, done in
                             ;delayed branch below
*
* Return
*
    LDI     *-FP(1),R1       ;load return address
    BD      R1              ;branch back
    LDI     *FP,FP          ;restore Frame Pointer
*** NOP
    STI     AR0,*+AR2(0)       ;restore circular buffer pointer
    SUBI    2,SP               ;Restore Stack Pointer
*** B       R1                 ;Branch occurs here
```

Example 3. C-calling shell

```
.
.
.
#include <kpstring.h>
.
.
.
/***********************************************************/
/*                MAIN()                                   */
/***********************************************************/
void main(void)
{
    float        *noise;
    KPSTRING     cs;
.
.
    makeNoise(&noise, 100, 2);
    createString(&cs, 50, noise);
    cs.sustainFactor = 0.49;
.
.
for(;;)
   {
.
.
.   if(NoteOn) initString(&cs, currPitchPer);
    string(&cs,output,blockSize);
    for(i=0;i<blockSize;i++) output[i] = amplitude*output[i];
.
.
.
    }
}
```