A Simple Voice Scrambler using the TMS320C6x DSK

Rulph Chassaing, Aaron Pasteris
University of Massachusetts Dartmouth

Abstract

A simple voice scrambling scheme is implemented on the TMS320C6211 DSK as an Undergraduate research project. The approach makes use of basic and simple algorithms for filtering and modulation. With voice as input, the resulting DSK output is scrambled voice. The original unscrambled voice is recovered when the DSK output is used as the input to another DSK running the same identical program.

Using an up-sampling scheme to process at a sampling rate of 16 kHz in lieu of the 8 kHz rate set on the DSK, a better performance allowing for a wider input signal bandwidth is achieved.

This project was implemented on the TMS320C31-based DSK[1] as a mini-project requirement in a Senior Elective course: Introduction to DSP (it was also implemented on the TMS320C25[2]) without using up-sampling. EE students at the University of Massachusetts Dartmouth implements this scheme with MATLAB in a Junior Linear Systems course.

Introduction

Digital signal processors are currently used for a wide range of applications from communications and controls to speech processing. They are found in cellular phones, fax/modems, disk drives, etc. They continue to be more and more successful because of the availability of low-cost support tools. DSP-based systems can be readily reprogrammed for a different application.

The C6x is Texas Instruments' (TI) highest performance processor based on the Very Long Instruction Word (VLIW) architecture. This type of architecture is very suitable for multitasking. The C6x supports a 32-bit address bus to address 4G bytes, and two sets of sixteen 32-bit registers. It contains eight functional/execution units composed of six ALU's and two multiplier units.

The internal program memory of the C6x is structured as "fetch packets" with a 256-bit instruction and a 256-bit bus. As a result, a word (VLIW) can be fetched every cycle. For example, a C6x with a 200 MHz clock is ideally capable of fetching eight 32-bit instructions, which forms a fetch packet, every 5 ns. On-chip memory is available as program cache, data cache, and RAM/cache. The exact amount and configuration of memory depends on the specific member of the C6x family of processors. For example, the fixed-point C6211 (which is on-board TI's popular

C6x DSK), has two Level-1 (L1) program and data cache of 4K Bytes each and a Level-2 (L2) 64K Bytes which can be used either as RAM or cache. The C6x internal data memory can be accessed as bytes, 16-bit (half-word) or 32-bit words.

The fixed-point C6211-based DSK is TI's lowest cost development system based on the C6x processor. The DSK board includes TI's 16-bit AD535 data converter, which contains an A/D and a D/A. The AD535 on board the DSK has a sampling rate of 8 kHz. It includes an antialiasing as well as a reconstruction filter. The C6x-based DSK is well suited as an educational tool. A DSK based on the floating-point C6711 is expected this year. The fixed-point C6211 is pin-compatible and upward ASM code compatible with the C6711 floating-point processor.

The DSK package includes the popular Code Composer Studio (CCS)[11] which provides an integrated development environment (IDE), bringing together the necessary support tools such as the assembler, C compiler, editor, debugger. A real-time data exchange (RTDX) capability with CCS, allows for the transfer of data between the PC host and the processor without stopping the application running on the target.

The syntax of the C6x assembly code is simpler than assembly code of the TMS320C5x or TMS320C3x processors. For example, such type of C3x instruction: DBNZD decrements a counter, then branches if not zero, with delay is not used with the C6x. On the C6x, the loop counter is decremented with a separate instruction and NOP's (no operations) are used to handle the delays associated with a branch instruction.

These "single-task" types of instructions on the C6x is a key factor in achieving a very efficient C compiler. It is relatively simpler to program the C6x in assembly code, compared to the C5x or C3x; however, optimizing such code can be quite challenging. One would need to resequence the order of the instructions in order to obtain as many as eight instructions in parallel (within one fetch packet), considering the delays associated with load, multiply, and branch instructions.

Linear assembly code provides a good compromise between C and assembly code. To program in linear assembly, one needs to know the syntax of the C6x instruction set but needs not specify the actual registers or functional/execution units to be used. Linear assmbly code is then a "cross" between C and assembly code and can in general be optimized to produce more efficient code than with C.

Implementation

The scrambling method used is commonly referred to as frequency inversion. It basically takes an audio range (represented by the band 0.3 – 3 kHz) and "folds" it about a carrier signal. The frequency inversion is achieved by multiplying (modulating) the audio input by a carrier signal, causing a shift in the frequency spectrum with upper and lower sidebands. On the lower sideband, which represents the audible speech range, the low tones are high tones, and vice versa

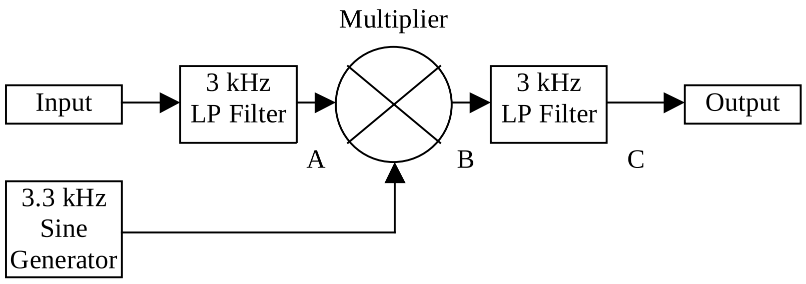

Figure 1. Block diagram of the scrambling scheme.

Figure 1 shows the block diagram of the scrambling scheme. Its attractiveness comes from its simplicity since only simple DSP algorithms are utilized (filtering, sine generation/modulation, and up-sampling). The input signal is first lowpass filtered and the resulting output (at point A) is multiplied (modulated) by a 3.3 kHz sine function with data values in a buffer in memory (look-up table). The modulated signal (at point B) is filtered again, and the overall output is a scrambled signal (at point C).

Using the resulting output as the input to a 2nd DSK running the same algorithm yields the original unscrambled input. Note that the program still runs on the 1st DSK when it is disconnected from the parallel port cable.

Figure 2 shows the C program to implement the voice scrambler. There are three functions besides the function main. One of the functions is to process the data. This function first calls another function "filter" so that the input signal is lowpassed for antialiasing. The resulting

```
// SCRAM16k_poll.c  Voice Scrambler Program
#include "C6211dsk.h"
#include "C6211dskinit.h"                    // DSK support file
#include "sine.h"                            // file with sine data values
#include "coeff.h"                           // coefficient file
short processdata(short data);
short filter(short inp,short *mem);
short MultSine(short input);
static  short filter1[COEFF],filter2[COEFF];
short input, output;

void main()
{
  int i;
  comm_poll();                                // init DSK/codec
  for( i=0; i< COEFF; i++)
```

```
 {
        filter1[i] = 0;                  // init buffer for filtered input signal - 1st filter
        filter2[i] = 0;                  // init buffer for modulated signal -  2nd filter
 }
 while(1)
 {
        input = input_sample();          // input data from codec
        processdata(input);              // process the sample twice to upsample
        output = processdata(input);     // and throw away the first result
        output_sample(output);           // to decimate, then output
 }
}

short processdata(short data)
{
        data = filter(data,filter1);     // call filter function – 1st filter
        data = MultSine(data);           // call function to generate sine and modulate
        data = filter(data,filter2);     // call filter function – 2nd filter
return data;
}

short filter(short inp,short *mem)      // implements lowpass filter
{
        int j;
        long acc;
        mem[COEFF-1] = inp;              // bottom memory for newest sample
        acc = mem[0] * coeff[0];         // y(0) = x[n-(COEFF-1)] * h[COEFF-1]
        for (j = 1; j < COEFF; j++)
        {
                acc += mem[j] * coeff[j];   // y(n) = x[n-(COEFF-1-j)] * h[COEFF-1-j]
                mem[j-1] = mem[j];          // update delay samples (x(n+1-i)=x(n-i)
        }
                acc = ((acc)>>15);          // scale result
                return acc;                 // return y(n) at time n
}

short MultSine(short input)              // sine generation and modulation function
{
   static int j=0;
   input = (input * sine[j++]) >> 11;    // input to multiplier * sine data
   if(j>=SINE) j = 0;
   return input;                         // return modulated signal
}
```

Figure 2. Program for implementing voice scrambling scheme.

output (filtered input) becomes the input to the multiplier/modulator. The function MultSine generates a 3.3 kHz sine using data values in a buffer representing a sine, then multiplies/modulates the filtered input signal with the 3.3 kHz sine data. This produces two sidebands. The modulated output is again filtered so that only the lower sideband is kept.

With voice as input, the overall filtered and modulated signal is scrambled voice. For example, with a 2-kHz input sinusoid, the resulting output is a lower sideband signal of 1.3 kHz (3.3 kHz – 2 kHz). With a sweeping input sinusoidal signal increasing in frequency, the resulting output is the sweeping signal "decreasing" in frequency.

A 2nd DSK is used to recover/unscramble the original signal (simulating the receiving end). Using the output of the 1st DSK as the input to the 2nd DSK and running the same program produces the reversed procedure yielding the original unscrambled signal. If the same 2-kHz original input is considered, the 1.3 kHz as the scrambled signal becomes the input to the 2nd DSK, and the resulting output is the original signal of 2 kHz (3.3 kHz – 1.3 kHz).

The up-sampling scheme to obtain a 16-kHz sampling rate is achieved by "processing" the data twice and retaining only the 2nd result. This allows for a wider input signal bandwidth to be scrambled.

A buffer in memory is used to store the filter coefficients arranged in memory as h[COEFF-1], …, h(0) where COEFF set to 113 (in coeff.h) is the order of the filter. Two other buffers are used for the delay samples, one for each filter, and are arranged in memory as x[n-(COEFF-1)], …, x[n], where x[n-(COEFF-1)] represents the oldest sample.

Conclusion

Interception of the speech signal can be made more difficult by dynamically changing the modulation frequency, and including, or omitting, the carrier frequency according to a predefined sequence. For example, a code for no modulation, another for modulating at frequency f_{c1}, and a third code for modulating at frequency f_{c2}.

This project can be readily duplicated at other institutions. Contact Rulph Chassaing at chassaing@msn.com for support files, etc.

Acknowledgement

Grants from the National Science Foundation (NSF) over the last few years provided the support to offer several workshops on Applications in DSP for many faculty over several years. The continued support of Texas Instruments is also appreciated.

Bibliography

1. R. Chassaing, Digital Signal Processing-Laboratory Experiments Using C and the TMS320C31 DSK, J. Wiley, 1999.
2. R. Chassaing and D. W. Horning, Digital Signal Processing with the TMS320C25, Wiley, 1990.
3. TMS320C6201/6701 Evaluation Module User's Guide, SPRU269, Texas Instruments, Inc., 1998.
4. TMS320C62x/C67x CPU and Instruction Set Reference Guide, SPRU189, Texas Instruments Inc., Dallas, Tx., 1998.
5. TMS320C62x/C67x Programmer's Guide, SPRU198, Texas Instruments, Inc., 1998.
6. TMS320C62x/C67x Technical Brief, Texas Instruments, Inc., Dallas, Tx.
7. TMS320C6x Assembly Language Tools User's Guide, SPRU186, Texas Instruments, Inc., Dallas, Tx., 1998
8. TMS320C6x Optimizing C Compiler User's Guide, SPRU187, Texas Instruments, Inc., Dallas, Tx., 1998
9. TMS320C6211 Fixed-point digital signal processor- Advance Information, SPRS073A, Texas Instruments Inc., Dallas, Tx., 1999.
10. TMS320C62xx Peripherals Reference Guide, SPRU190, Texas Instruments, Inc., 1997.
11. Code Composer Studio Tutorial, SPRU301, Texas Instruments, Inc., Dallas, Tx., 1999.
12. W..J. Gomes III and R. Chassaing, " Filter Design and Implementation Using the TMS320C6x Interfaced with MATLAB," in Proceedings of the 2000 ASEE Annual Conference.