Simulation of Voice Processing Applications through VLIW DSP Architectures

N. Sedaghati-Mokhtari[1], M. N. Bojnordi[1], A. Farmahini-Farahani[1], M. Mousavinezhad[2], and S. M. Fakhraie[1]
[1] Silicon Intelligence Laboratory, School of ECE, University of Tehran, Tehran 14395-515, Iran
[2] Iran Telecommunication Research Center (ITRC), Tehran 14155-3961, Iran
{n.sedaghati,m.bojnordi,a.farmahini}@ece.ut.ac.ir, m.musavi@itrc.ac.ir, fakhraie@ut.ac.ir

***Abstract*— This paper presents simulation of G.729a speech codec, as a compute-intensive voice processing application, on a VLIW DSP architecture. The simulator of target processor is verified by the Texas Instruments C62xx VLIW DSP architecture. The cycle-accurate simulator is implemented using C++ programming language. The G.729a reference code is modified for single- and multi-channel voice coding and is executed in the simulation environment. The experiments are performed for two modes: single and multiple channels of voice data. The obtained results demonstrate functionality of the VLIW DSP architecture for efficient evaluation of single- and multi-channel voice processing. It is achieved that the target processor is functionally available to support real-time execution of up to 10 voice channels at the 200 MHz working frequency while considering all of the operating conflicts.**

I. INTRODUCTION

Simulation tools are essential aides to both designers and researchers in computer architecture, due to their ability to study and validate new designs without the cost of actually building the hardware. They also provide for a good platform on which researchers can explore a wide range of design choices, which might be practically not feasible. Simulation tools also allow one to study the combined interaction of all the architectural features, before anything is built and can bring out potential bugs that might not have otherwise been detected [1].

In this work, we present simulation of G.729a speech codec [2], a sample voice processing application, through VLIW DSP architecture. A cycle-accurate simulation environment is developed for the target VLIW processor. The simulator architecture is verified by Texas Instruments C62xx DSP processor using C++ programming language. We present the multi-channel modification for reference code (i.e. reentrancy) and its effects on the simulation of multi-channel voices.

The rest of the paper will be as follows: Section II discusses the simulator architecture plus the review on cycle-accurate implementation concerns. A brief overview on the G.729 speech codec is presented in Section III. Multi-channel voice simulation and required techniques for code-modification will be described in Section IV. Experiments and concluding words will be presented in the Sections V to VI.

II. SIMULATOR ARCHITECTURE

We developed a cycle-accurate simulator for VLIW DSP architectures. The model is a high-level implementation with reverse execution mechanism for pipeline modeling. It has been designed and implemented based on Texas Instruments (TI) C62xx DSP processor [3][4]. In order to execute real bit-accurate DSP operations of various bit widths, the model employs a specific type of data as a C++ class which is called *DSPDT*. *DSPDT* is a class consisting of various data fields which simulate real DSP behaviors. It also consists of methods which are divided to two categories in terms of functionalities: simulator methods and processor methods. Simulator methods perform functions required for simulator implementation while the processor methods implement the target processor's instruction set (in this work C62xx instruction set). *DSPDT* provides a bit-accurate data type for modeling the DSP operations in C++ language [5].

We choose C++ for implementing the processor model and also the simulation environment. Sequential nature of this programming language results in many challenges such as real pipeline modeling, clocking strategy and shared resource management. In order to tackle these problems, simple message passing strategy and updating mechanism are considered. The processor simulator, in addition to datapath and controller, consists of memory, register file, and statistical reporting and monitoring (SRMU) units. The *SRMU* is responsible for gathering all the user-requested information from the simulation. This unit is interfaced to standard file system to store the monitoring and statistical reports in the desired files [5].

Using CCStudio and MATLAB toolsets, the processor model and simulator are validated by executing two kinds of DSP benchmarks, single- and multiple-VLIW-packet programs. The benchmarks are used for validating the individual block functionalities and the whole Datapath architecture of the simulator, respectively. Multi-packet programs are DSP microbenchmarks such as IIR filter, correlation, and FFT blocks [5].

1-4244-1378-8/07/$25.00 ©2007 IEEE.

III. G.729A SPEECH CODEC

G.729 is an audio data compression algorithm for voice that compresses voice audio in chunks of 10 milliseconds. Music or tones such as DTMF or fax tones cannot be reliably transported with this codec, and thus use G.711 or out-of-band methods to transport these signals. G.729 is mostly used in Voice over IP (VoIP) applications for its low bandwidth requirement. Standard G.729 operates at 8 kbit/s, but there are extensions, which provide also 6.4 kbit/s and 11.8 kbit/s rates for marginally worse and better speech quality respectively. Also very common is G.729a which is compatible with G.729, but requires less computation. This lower complexity is not free since speech quality is marginally worsened. The annex B of G.729 is a silence compression scheme, which has a VAD module which is used to detect voice activity, speech or non speech. It also includes a DTX module which decides on updating the background noise parameters for non speech (noisy frames). These frames which are transmitted to update the background noise parameters are called SID frames. A comfort noise generator (CNG) is also there, because in a communication channel, if transmission is stopped, because it's not speech, then the other side may assume that link has been cut [2].

IV. MULTI-CHANNEL VOICE SIMULATION

Targeting efficient simulation of the voice processing applications, we employ G.729a [2] standard reference code which is developed and released in C language. We modify the code for real DSP execution. The code is modified, in terms of interfaces to files, to be compatible to real DSP architectures. Also, it compiled for target processor using CCStudio code generation tool. The disassembly code is parsed to be readable by the simulator interface model. The sample data are also put into the files and read by the data interface model of the simulator. It should be noted that the code is a standard code which is not obviously the optimized one, but perform the simulation requirements of the application evaluation.

For multi-channel implementation of a sample DSP application, the code should be significantly modified. As an important modification, it must support code reentrancy. The code is reentrant if it is able to be executed on multiple contexts without any changes in the program structure [6]. In this way, for each channel there is two parts of data: a single memory space for private variables (private context) and a shared space for shared variables between multiple channels. Major concern of the reentrant code designer is eliminating global static variables. The global static variables caused overwriting of values after execution of multiple channels with different contexts.

For example, a filter function with global static variables is shown in Fig. 1 (a). In this example, x_0 and x_1 are defined as global static variables. Therefore they are commonly accessed by various parts of the code at the run time. As a result, many unexpected errors may be occurred when a new data is processed for the new channel. In order to tackle this problem, all global static variables are defined as an array of elements. This way, a channel pointer is used for accessing corresponding set of variables to the desired channel in the array. Hence, the function can be rewritten as shown in Fig. 1 (b).

The filter code can be presented more structured as shown in Fig. 1 (c). By this modification, an efficient data type is used for maintaining state of each channel. Such reentrant code generation strategy is recommended for all parts of the codec program to manage data memory efficiently. Generally, data memory is divided into three parts:

- *Context data*: The context data includes static variables and arrays which preserve their content during exchanging from one frame to another in a desired channel. Context data for G.729 are stored in two specific data structures, namely ENCODER_G729_MEM_BLOCK and DECODER_G729_MEM_BLOCK. Sizes of these memory elements are 1902B and 1310B, respectively.
- *Table*: It contains constant values used for both encoder and decoder. G.729 uses the data structure G729_TABLES for this goal which occupies data memory of size 6024B. As mentioned tables are constant during exchanging the channels and frames. In other words, these variables do not affect code reentrant.
- *Local variables and arrays*: These variables are allocated on the stack up to 2780B. Like the tables, local variables do not affect nor affected by the code reentrant; however, they are varied during execution frequently.

```
int x0,x1;
void filter (int input[] , int length){
  int i,temp;
  for(i = 0;i < length; i++){
    Temp = input[i] - x0 + (10 * x1 + 20) / 2;
    x1 = x0;
    x0 = input[i];
    input[i] = temp;
  }
}
```

a)

```
void filter (int input[] , int length, int *x){
  int i,temp;
  for(i = 0; I < length; i++){
    temp = input[i] - x[0] + (10 * x[1]+ 20) / 2;
    x[1] = x[0];
    x[0] = input[i];
    input[i] = temp;
  }
}
```

b)

```
typedef struct FILTER_state{
  int x0;
  int x1;
}FILTER_state;
void filter (FILTER_state *delay, int input[] , int length ){
  int i,temp;
  for(i = 0; I < length; i++){
    temp = input[i]- delay->x0 + (delay->x1 * 10 + 20) / 2;
    delay->x1 = delay->x0;
    delay->x0 = input[i];
    input[i]=temp;
  }
}
```

c)

Figure 1. Sample filter code: a) static global variables, b) with arrayed variables and c) structured variables.

In addition to the required modification for static variables, data interface unit should be prepared and modified to support multi-channel data entrance.

V. EXPERIMENTAL RESULTS

The obtained results of G.729a speech codec for 10ms (one frame) single-channel voice data is presented in TABLE I. [5]. Although the code is generic and obviously non-optimized; but it satisfies the simulation requirements for application evaluation. After preparing the code by applying the mentioned modifications, using CCStudio has been compiled and executed. Results of executing the multi-channel code are listed in Table II.

Presented results in this table are obtained for processing frames (10 milliseconds) of multiple parallel channels. During this time period, number of simulation cycles and executed instructions are counted. Assuming 200 MHz working frequency for target processor, the simulator model is able to execute up to 1600 million instructions per second. Assuming 2 million cycles for processing of each voice frame, maximum number of simulation cycles for real-time processing demonstrates that the processor is able to process up to 10 voice channels in a typical real-time application. In order to process 32 voice channels, working frequency of 1GHz is needed. Also, simulation results indicate that modification and preparation of the codec program resulted in 18% code increase. In addition, 40KB and 110.5KB of data memory are required for real-time processing of 10 and 32 voice channels, respectively. This limitation is imposed because of the inefficiency of the application code, many frequent memory accesses and context-switching overheads during multi-channel operations.

VI. CONCLUSION

Simulation of G.729a speech codec using a cycle-accurate simulation environment in single and multi-channel modes is presented in this paper. The simulator architecture is verified by a VLIW DSP processor family, Texas Instruments C62xx DSP processor using C++ programming language. The simulator is functionally validated by real DSP benchmarks using CCStudio and MATLAB toolsets. Simulation results indicate that the model supports real-time G.729a speech coding of 10ms single-channel voice by about 115 kilo-cycles and 313 kilo-instructions. Also, it demonstrates that the target DSP processor is able to process G.729a coding of up to 10 real-time voice channels.

TABLE I. SIMULATION OF SINGLE-CHANNEL VOICE.

Measure	Instruction Count	Cycle Count	Code Size (Word)
Encoder	213526	81565	77395
Decoder	90541	34823	35392

TABLE II. SIMULATION OF MULTI-CHANNEL VOICE.

No. of voice channels	Instruction Count	Cycle Count
2	705739	261788
3	1111060	412183
4	1692323	627725
5	2205778	818242
6	2915166	1081356
7	3324859	1233328
8	4100496	1521044
9	4613951	1711506
10	5191146	1925611
11	5864132	2175249
16	9523366	3532611
32	25981373	9637569

ACKNOWLEDGEMENT

N. Sedaghati-Mokhtari wishes to express his gratitude to Iran Telecommunication Research Center (ITRC) for financial support of his research.

REFERENCES

[1] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta "Complete computer system simulation: the SimOS approach," *IEEE Parallel and Distributed Technology: Systems & Applications*, vol. 3, no. 4, pp. 34-43, Winter 1995.

[2] *Coding of speech at 8Kbps using the Conjugate Structure Algebraic Code Excited Linear–Prediction (CSACELP)*, ITU-T Draft Recommendation G.729, June 1995.

[3] J. Turley and H. Hakkarainen. "TI's new 'C6x DSP screams at 1,600 MIPS," *Microprocessor Report*, vol. 11, no. 2, February 1997.

[4] *TMS320C6000 CPU and Instruction Set Reference Guide*, Texas Instruments Literature Number SPRU189C, March 1999.

[5] N. Sedaghati-Mokhtari, M. N. Bojnordi, A. Hormati, and S. M. Fakhraie, " Efficient simulation of VLIW DSP processors," *IEEE International Conference on Signal Processing and Communications*, , Dubai, November 2007, *in press*.

[6] *G.729/A Speech Coder: Multichannel TMS320C62x Implementation*, Texas Instruments Literature Number SPRA564B, February 2000.