MDST: Multiprocessor DSP Simulation Toolkit for Voice Processing Applications

Naser Sedaghati-Mokhtari, Mahdi Nazm Bojnordi, and Sied Mehdi Fakhraie
Silicon Intelligence and VLSI Signal Processing Laboratory,
School of ECE, University of Tehran, Tehran 14395-515, Iran
{n.sedaghati,m.bojnordi}@ece.ut.ac.ir, fakhraie@ut.ac.ir

Abstract— **In this paper, we propose a multiprocessor DSP simulation toolkit suitable for performance evaluation of data-parallel applications like voice processing. The proposed toolkit uses the benefits of multi-level parallelism and clustering. Different DSP clusters are considered for the multiprocessor DSP simulation engine in which the DSP processors are grouped to cooperate. Satisfying the communication requirements, two global and local communication engines (GCE and LCE) implement the real behavior of intra- and inter-cluster communications. Using efficient abstraction levels for interconnections reduces the simulation time significantly. Abstract communication modeling, cycle-accurate behavior, and multi-level controlling are the most important features of the proposed simulation platform. Performance of the simulator is verified by standard single- and multi-channel voice processing applications such as ITU-T G.729a speech codec.**

Keywords- MDST, Multiprocessor DSP Simulation toolkit, Voice processing applications

I. INTRODUCTION

Architecture designers tend towards harnessing many processing cores instead of increasing the chip frequency to achieve higher performances. For example, in 2004 Intel announced it is moving its PC processor development towards multiprocessor solutions and away from a sole focus on increasing chip frequency. Also, in the field of digital signal processing, multiprocessor DSP architectures become compelling solutions for high-performance applications such as packet telephony, 3G wireless infrastructure, and WiMAX [1]. Therefore, the designers proposed many multiprocessor DSP systems [2]-[8].

High quality voice communications is achieved by using an IP network that have low delay, low jitter, low packet loss, and high reliability even during congestion conditions [9]. Voice processing applications such as speech codecs and echo cancellation are major domain in which highest computational power is a dominant requirement. Multiprocessor DSP systems are the most efficient source of computation for such systems. Industry vendors such as Texas Instruments, Zarlink, MindSpeed, Cradle, and Freescale have released multiprocessor DSP solutions for voice applications [10]-[14].

Like single core architectures, multiprocessor systems require suitable and reliable tools and environments in all levels of system design and verification. This way, simulation environments are recommended by the designers to address the real design issues. In the literature, there are many multiprocessor simulators with performance modeling and simulation techniques ranged from several parallel computers down to chip multiprocessors. Each of them particularly targeted with their existed architectural issues. Existing simulation environments are like that SimpleScalar [15], SimOS [16], SimICS [17], RSIM [18], and ML-RSIM [19]. Realizing the performance, accuracy, and limitations of simulators are critical issues during selecting a tool to be employed for desired research questions. Especially in the domain of multiprocessor systems for digital signal processing applications there are only few existing simulators that are restricted to the underlying architecture. An instance of such simulators is software developed for Daytona multiprocessor architecture [20]. These software tools are specially developed for a specific architecture; hence, they cannot consider all required aspects of simulation and design. Therefore, a generalized simulation platform which targets the special DSP application still remains as a necessary tool for researchers.

This paper proposes a multiprocessor DSP simulation toolkit for evaluating performance of different data-parallel applications. Rest of the paper is organized as follows. In Section II, the simulation toolkit including all implemented components, application mapping and scheduling will be described. Experimental results and conclusion are presented in Section III and IV, respectively.

II. SIMULATION TOOLKIT

Simulation of a high-performance voice processing application requires a platform which supports multiple autonomous and powerful processing engines and also requires specific communication engines for efficient program and data transfer. Addressing these issues, this paper proposes MDST as a multiprocessor DSP simulation toolkit for voice processing applications. Overall block diagram of MDST system consists of the blocks shown in Fig.1. In addition to the multiprocessor DSP model, following units are defined and implemented: User Interface Unit (UIU), Application Mapping and Task Scheduling Unit (AMTSU) and Statistical Reporting and Monitoring Unit (SRMU).

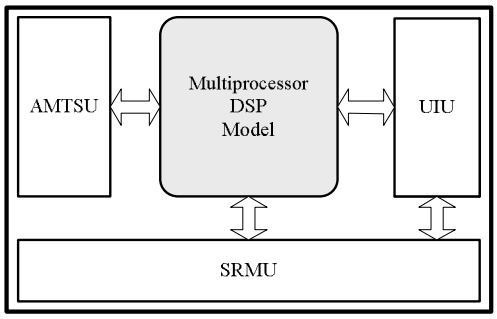

Figure 1. MDST: Multiprocessor Simulation Toolkit

A. *Multiprocessor DSP Model*

Inside MDST a multiprocessor DSP model plays an important role for simulating the real behavior of processing environment. The system consists of several parts: memory modules (global and local), communication engines (global and local), DSP cores and interfaces. The proposed system provides a multi-cluster environment in which every cluster is composed of a multiple autonomous DSP cores. Communication engines are responsible for all kinds of global and local transfers between memory and processing modules. Each communication engine consists of: *abstract model of interconnection*, *controller*, and *interfaces*. The proposed model supports two levels of communications: global and local. From the viewpoint of DSP clusters, global and local communication engines handle the inter- and intra-cluster traffics, respectively.

Fig. 2 demonstrates the overall block diagram of the multiprocessor DSP simulation model. At the center of the model, global communication engine (GCE) performs the inter-cluster traffics (program, data, and control transfers) between global memory, protocol processor, I/O system and DSP clusters through special interfaces. The system consists of two types of interfaces: protocol processor and I/O subsystem. Protocol processor interface is a starting point for generating all the controlling signals which must be applied to other parts of the platform, according to application behaviors.

The memory interface (PI and DI) provides different kinds of program and data transfers between memory and other requesting/granted components such as DSP clusters. Each cluster is communicated with the global system through three program (PI), data (DI) and control (CI) interfaces. In order to have a suited top-level controlling, a global controller (GC) is proposed managing inter-cluster transfers. From the application perspective, all DSP-dependent parts of the user application (program/data transfers and computation tasks) are directly controlled by the GC module. GC handles two kinds of transfers: top-down (program/data read) and bottom-up (data write).

1) *DSP Core Simulation Model*

In order to setup a multiprocessor DSP modeling environment, a DSP core model is needed as a cycle-accurate computational model. The core model in [27] is a cycle-accurate VLIW DSP model employing a specific pipeline modeling technique called reverse execution. It has been verified by the Texas Instruments C62xx [21] processor architecture. We develop the simulator using C++ programming language. For bit-accurate implementation of signal processing operations, we design and implement a DSP-specific data type, called DSPDT. The simulator handles pipeline resources (memories and register files) during concurrent accesses by an updating method.

The simulator consists of several numbers of classes familiar to the hardware modeling concepts. The hierarchical design for classes makes the coding and implementation of the simulator easier and less complicated. Each block of a pipeline stage is considered as an independent class with its own methods and attributes.

Fig. 3 shows the DSP core pipeline stages regarding the data flow. Each block shown in this figure is an object instanced from a corresponding C++ class. While each class is model of a sub-module of DSP core. Blocks in each stage, only use outputs coming from the predecessor stage in order to generate the output values.

For the DSP core, four major functions are considered during each round of the simulation loop.

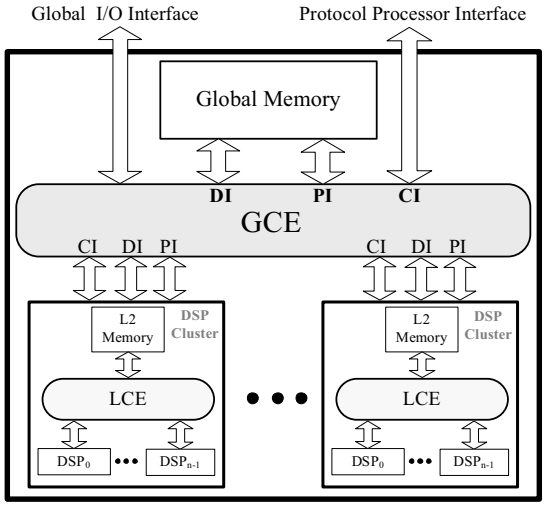

Figure 2. Multiprocessor DSP model

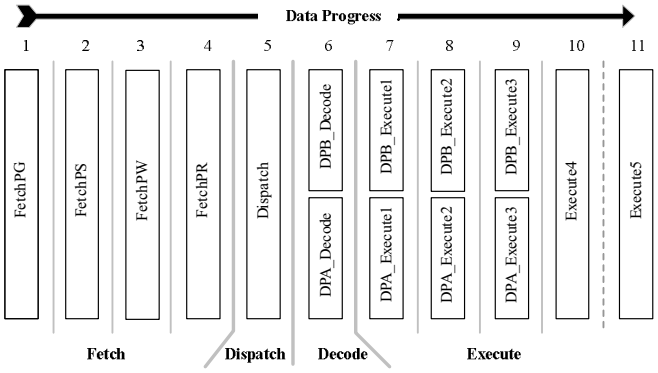

Figure 3. Pipeline stages of the DSP core model

1) Provide back up of the controlling and state signals which are use by the blocks. This is necessary to preserve them for using as inputs to every desired block.
2) Call *run* function of all blocks in reverse order of the pipeline data flow. Reverse order must be considered to preserve output of stages until they are used by the successors.
3) Synchronize all data with global clock and capture in the sequential parts. This step is considered as register/memory updating, as well.
4) Provide execution statistics and internal state reports.

Each DSP core in a cluster environment follows some specific operations. The DSP core state machine depends to the core pipeline and cluster controlling modes. Fig. 4 shows required finite state machine (FSM) for controlling each DSP core. The FSM follows a cycle-accurate procedure of signal management and each transition takes place after several delay elements (simulation cycles).

Before the core simulation started, at the start stage, the DSP core is initiated by the applied configuration parameters. The configurations are applied from cluster controller and mainly affect the core simulation. The parameters have no changes on the pipeline stages and core architecture. After core configuration, four independent procedures may be happened during the core operation: program read (pr), data read (dr), data write (dw), and core process (cp). These operations construct 4-bit transition signal in this order: (pr-dr-dw-cp). From the simulation rules, for each DSP core model:

- The program and data transfers (read or writes) could not be occurred concurrently.
- Data transfer only happens once at time (read or write).
- There is no overlapping between core processing and program/data transfer in a single core.

According to the mentioned rules, there are several undefined states which are not shown at the Fig. 5 FSM is changed from one state to another according to the defined 4-bit input. For example, being in the **core process** state, input value 0100 (i.e. pr(0), dr(1), dw(0), cp(0))direct the FSM to **data read** state. When none of four mentioned operations are performed, the FSM finishes the processing.

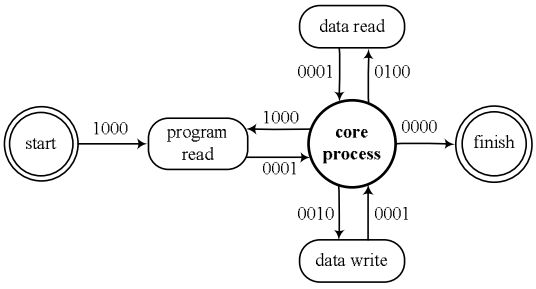

Figure 4. FSM for Controlling a DSP core

2) DSP Cluster

A DSP cluster is composed of DSP cores, cluster memory (L2) and local communication engine (LCE). LCE connects DSP cores to the cluster memory module through a cluster interconnect model and cluster controller. Fig. 5 shows block diagram of the DSP cluster model beside the system components.

The model consists of three different communication interfaces which are specified for each DSP core, as follows:

- *Control Interface (CI)*: Manages the transfer of controlling signals between DSP cores and the cluster controller.
- *Data Interface (DI)*: Is used for DSP cores to handle their local read/write operations from/to L2 memory (cluster memory).
- *Program Interface (PI)*: Each DSP core used this interface for handling the program read operations from L2 memory.

In general, LCE performs the following transfer operations:

- Intra-cluster read/write transfers between cluster memory and DSP cores,
- Inter-cluster memory transfers between global and cluster memories using the cluster controller.

According to the controlling mechanism inside each DSP core, the cluster controller manages all cluster communications regarding the FSM shown in Fig. 6.

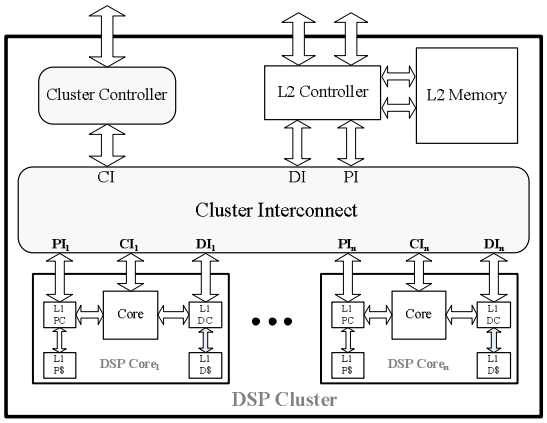

Figure 5. DSP cluster model

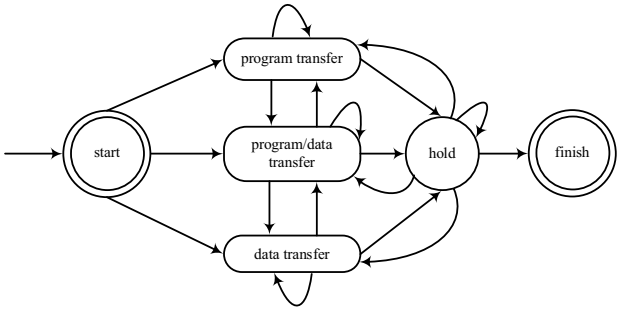

Figure 6. FSM of the cluster controller.

As depicted in this figure, the following states are defined for cluster controller (CC):

- **Start**: This is an initialization state for preparing the simulation environment affected by two kinds of configuration parameters: *application-oriented* and *user-defined*. Application parameters are extracted from application and applied to the system components. Such configuration is applied to both GCE controller and memory unit. GCE controller's configuration hierarchically affects low-level components such as clusters and LCE controllers. Another type of configuration is user defined and applied to several components such as DSP clusters, LCE controller.
- **Program transfer**: CC goes to program transfer state when program requests arrive. If the program code exists in L2 memory, CC coerces the L2 controller to the control the requested transfer. Otherwise, the CC provides an inter-cluster transfer from global memory to the desired DSP cluster. While completing the program transfer for a block code, other program requests await in the pending state (in separated queues). Both program and data transfers are allowed in the same simulation cycle.
- **Data transfer**: In contrast to the program, data transfer occurs in two directions: from DSP to global memory (write) and from global memory to DSP cores (read). Only one transfer occurs in a simulation cycle because of limitation in data addressing and transfer controlling.
- **Program/data transfer**: For simultaneous transfers of data and program code, the controller is transited to this state when both data and program transfer requests are arrived from the cluster. When each of the program or data transfers is completed, the controller leaves this state to the single transfer states: program or data transfer.
- **Hold**: This state is reached if the controller becomes idle while no request from DSP cores is delivered. In this state, CC manages non-transfer controlling tasks for DSP cluster. Also, in the case of any request, the controller transits to the proper transfer states.
- **Finish**: The controller reaches this state at the end of simulation time or when the GC indicates the termination for the cluster.

When the program is transferred from global memory to the DSP cores, the broadcasting and multicasting capabilities are applicable. By checking the current pending requests, the cluster controller sends program to requesting DSP core and all other requesting cores with the same requests. Such capabilities significantly reduce the response time for multiple-transfer when there are equivalent requests in the pending states.

B. User Interface Unit (UIU)

User of MDST is provided by several simple simulation facilities. Configuring the simulation environment and also providing initializations for the system is performed using the user interface. The MDST user communicates with the simulation core using input (instruction for simulation) and output (information from simulation) interfaces provided by a GUI program. The unit is also interfaced with the multiprocessor DSP simulation model for basic configurations. At the end of simulation, the collected statistical simulation information is provided by SRMU.

C. Application Mapping and Task Scheduling unit (AMTSU)

In order to handle the application mapping and task scheduling issues, the MDST system employs a data parallel programming model to support efficient implementation of data-parallel algorithms.

1) Application Mapping

Fig. 7 shows structure of a telecommunication application mapping in the proposed simulation model. According to this figure, a telecommunication application covers both DSP- and protocol-oriented parts. For example, packetization and depacketization in voice over IP applications had better to be handled by a powerful RISC processor. According to the nature of data-parallel programming model, the environment needs to be provided by specific controlling mechanisms.

The proposed procedure maps non-protocol-related tasks to the MDST model in the proposed platform. The proposed mapping strategy, shown in Fig. 7, demonstrates how to map the tasks to the related implementation modules. DSP-related tasks are divided to two major parts: computation and communication. The computation part is indirectly mapped to the DSP cores. On the other hand, communication tasks are directly mapped to the communication engines (GCE and LCE). These two parts are not completely separated because they are handled concurrently during the MDST operation steps. Distinction between computation and communication decrease the complexity of the application mapping in the programming model. As mentioned in the figure, the MDST model is also concerns about the DSP-related tasks and only interface with the protocol-related part via specific interface module.

2) Task Scheduling

Scheduling the DSP part of the telecommunication application, as a first solution, thread-level or task-level partitioning are proper volunteers selecting most efficient parallelizable block codes of the application and assign them to the functional units for software-execution. For applications which are not inherently parallelizable, this approach injects task-level dependencies in terms of execution and data.

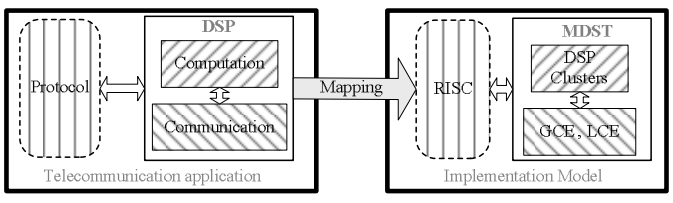

Figure 7. Mapping telecommunication application to the MDST

Execution dependencies increase latencies (a pipeline structure) while data dependencies increase implementation overheads such as capabilities for shared-memory coherence and consistency. Also, task-dependencies make completion of total program execution dependent on the completion of all the tasks mapped to the functional units.

For data-parallel applications, such as voice processing, where multiple independent flows (channels) of data are concurrently processed, the alternative solution is data-level partitioning. In this approach, in contrast to the former one, each channel-processor (DSP core) is executing the whole program code image for a particular data channel. Therefore, each DSP core completes its own execution of program code independent of other cores. The data-level partitioning is a fundamental concept proven in the simulation model. **Error! Reference source not found.**Fig. 8 shows sample data-parallel scheduling and code distribution for a cluster with 4 DSP cores.

According to the DSP core characteristics, addressing range and cache sizes, a program code is partitioned into several block codes (BC). There is only one real copy of the code in the main memory, but each DSP core is virtually scanning the whole program code in the terms of blocks to complete the code execution. Data-dependencies in an execution of a code cause changing the executing BC in each DSP core. As shown in this figure, since each DSP core requests for the next required BC, it is not needed for each DSP core to keep the whole code in the local memory. Therefore, in a simulation cycle, each DSP core executes its related BC. If more than one DSP cores request for the same BC, and when a program code is read, the corresponding BC will be broadcasted to the requesting units. This broadcasting feature in program distribution is significantly degraded the rate of redundant BC transfers in the communication engines.

III. EXPERIMENTS

Coding of speech signals is a necessary requirement in mobile communication networks, IP telephony systems, and VoIP applications. Several different coding and decoding algorithms have been developed and standardized. The most common speech codecs used are the ITU-T G.723.1, G.728, and G.729 for VoIP applications, and GSM Full- Rate (FR), Half-Rate (HR), and Enhanced Full-Rate (EFR) for mobile communications. Speech coding algorithms are usually computationally intensive and need a significant amount of signal processing power. Conventionally, either special-purpose DSP processors with codec-customized architectures have been designed to perform the speech signal processing tasks [22] or general-purpose DSP processors with highly optimized codec software [23]-[26] have been suggested.

This work chooses the second solution by experiencing the effects of computational power growth through the multiprocessor DSP platform. In order to experience the maximum processing power by a single DSP core [27], we demonstrate the G.729a execution results in TABLE I.

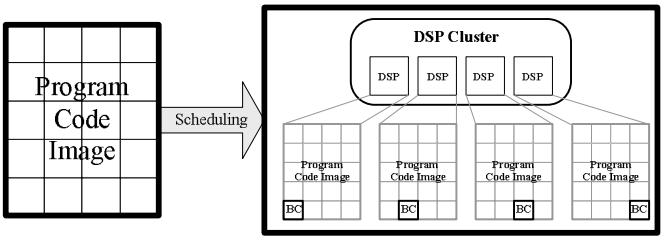

Figure 8. Sample data-parallel scheduling for a cluster with 4 DSP core.

TABLE I. ACHIEVED PERFORMANCES FOR G.729A SPEECH CODEC ON SINGLE-CORE DSP MODEL

Measure	Instruction Count	Simulation Cycles
Encoder	213526	81565
Decoder	905411	34823

The obtained results show that a single DSP core model supports up to 10 real-time voice channels while theoretically the architecture should be able to process more real-time voice channels [27]. This limitation has been imposed for the sake of inefficiency of the application code, many frequent memory accesses and context-switching overheads during multichannel operations.

According to achieved single-core DSP results, and also extreme achieved performance for single-core DSP systems with highly-optimized program code for speech codec applications [23]-[26], there is needed more parallelism by increasing the number of computational units (DSP cores). TABLE II. and TABLE III. show achieved performances by MDST for different number of DSP cores in single- and multiple-channel conditions. We remember that the DSP core model is compatible to [21] and considered to be operable in the real implementation working at 200 MHz with maximum performance of 1600 MIPS.

According to the obtained results, during G.729 speech coding on the MDST, 22% and 71% of the maximum available processing performance is consumed for DSP related tasks in single- and multiple-channel modes, respectively. Employing multichannel speech coding results in more efficiency and harnessing more processing power. Single-core simulation supports up to 10 voice channels while multi-core DSP degrade the performance of each DSP core down to 8 voice channels. This performance degradation is because of communication overheads in each DSP cluster and also waiting delays for program/data transfers.

TABLE II. MULTIPROCESSOR DSP PERFORMANCES VIA DIFFERENT NUMBER OF CORES AND CLUSTERS FOR SINGLE-CHANNEL G.729A CODEC

No. of DSP clusters	DSP cores per cluster	Maximum performance (MIPS)	Achieved Performance (MIPS)
1	2	3200	709
1	4	6400	1236
2	2	6400	1385
2	4	12800	2736
3	2	9600	2152
3	4	19200	3976

TABLE III. MULTIPROCESSOR DSP PERFORMANCES VIA DIFFERENT NUMBER OF CORES AND CLUSTERS FOR MULTIPLE-CHANNEL G.729A CODEC

No. of DSP clusters	DSP cores per cluster	No. of voice channels for each DSP	Maximum performance (MIPS)	Achieved Performance (MIPS)
1	2	2	3200	437
1	2	4	3200	1330
1	2	8	3200	2163
1	2	16	3200	6126
1	4	2	6400	825
1	4	4	6400	1807
1	4	8	6400	6422
1	4	16	6400	10392
2	2	2	6400	922
2	2	4	6400	2819
2	2	8	6400	4573
2	2	16	6400	12986
2	4	2	12800	1737
2	4	4	12800	3804
2	4	8	12800	8949
2	4	16	12800	21876

IV. CONCLUSION

A multiprocessor DSP simulation toolkit for efficient performance evaluation of data-parallel applications, especially voice processing, has been presented. The proposed toolkit uses the benefits of multi-level parallelism and clustering of the DSP cores. Different clusters are considered for the multiprocessor DSP simulation engine in which the DSP processors are grouped to cooperate together. In order to meet the communication requirements, two proper global and local communication engines (GCE and LCE) implement the real behavior of intra- and inter-cluster communications. Employing various levels of abstraction for interconnections reduces the simulation time significantly. Abstract communication modeling, cycle-accurate behavior, and multi-level controlling are the most important features of the proposed simulation platform. Performance of the simulator has been verified by the standard voice processing applications such as ITU-T G.729a speech codecs. Simulation results indicate that for single- and multiple-channel voice processing in G.729a speech codec, 22% and 71% of the DSP performance is consumed for computation goals, respectively.

REFERENCES

[1] I. Scheiwe, “The shift to multiprocessor DSP solutions”, online available at: http://www.dsp-fpga.com/articles/scheiwe/, retreived April 2007

[2] T. P. Barnwell, V. K. Madisetti, and S. J. A. McGrath, “The Georgia Tech digital signal multiprocessor”, IEEE Trans. on signal processing. vol. 41. no. 7, July 1991

[3] Srinath V. Ramaswamy and Gerald D. Miller, “Multiprocessor DSP architectures that implement the FCT based JPEG still pictures image compression algorithm with arithmetic coding,” IEEE Transactions on Consumer Electronics, vol. 39, no. 1, Feb. 1993

[4] H. Igura, S. Narita, Y. Naito, K. Kazama, c. Kuroda, M. Motomura, M. Yamashina, “An 8OOMOPS 11 OmW 1.5V parallel DSP for mobile multimedia processing,” ISSCC , pp: 292-293, CA, Feb. 1998.

[5] B. Ackland, A. Anesko, D. Brinthaupt, S. J. Daubert, A. Kalavade, J. Knobloch, E. Micca, M. Moturi, C. J. Nicol, J. H. O'Neill, J. Othmer, E. Säckinger, K. J. Singh, J. Sweet, C. J. Terman, and J. Williams, “A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP”, JSSC, vol. 35, no. 3, pp: 412-424, March 2000

[6] W. Zubin, T. Jun, W. Xiutan, and P. Yingning, “The structure and application of a new multi-DSP parallel computing architecture based on ADSP-2106x,” CIE International Conference on Radar, pp: 590-594, China 2001.

[7] G. Zhong, F. Xu, and A. N. Willson, “A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring,” JSSC, vol. 41, no. 2, pp: 483:495, Feb. 2006

[8] OMAP product bulletin, Texas Instruments, available at: www.ti.com

[9] J. Davidson, J. Peters, B. Gracely, Voice over IP Fundamentals, Cisco Press, 2000

[10] TNETV product series, Texas Instruments, available at www.ti.com

[11] Zarlink’s VoIP solutions, technical note, available at www.zarlink.com

[12] Comcerto product series, MindSpeed, available at www.mindspeed.com

[13] CT3600 MDSP Family, Cradle technologies, available at www.cradle.com

[14] MSC8144 multi-core DSP, Freescale semiconductore, available at www.freescale.com

[15] T. Austin, E. Larson, D. Ernst, “SimpleScalar: an infrastructure for computer system modeling,” IEEE Computer, vol. 35, pp 59-67, 1998.

[16] M. Rosenblum, E. Bugnion, S. Devine, S. A. Herrod, “Using the SimOS machine simulator to study complex computer systems,” ACM TOMACS special issue on Computer Simulation, pp: 79-103, 1997.

[17] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, “SimICS: a full system simulation platform”, IEEE Computer, vol. 35, no. 2, pp:50–58, Feb. 2002.

[18] V.S. Pai, P. Rangnathan, S.V. Adve, “RSIM reference manual, version 1.0”, tech. report 9705, Department of Electrical Computer Engineering, Rice University, Houston, TX, 1997.

[19] L. Schaelicke, M. Parker, “The design and utility of the ML-RSIM system simulator”, Journal of Systems Architecture, vol. 52, no. 5, pp 283-297, May 2006.

[20] A. Kalavade, J. Othmer, B. Ackland, K. J. Singh, “Software environment for a multiprocessor DSP”, ACM/IEEE Conference on Design Automation (DAC), pp: 827-830, Louisiana, 1999.

[21] J. Turley and H. Hakkarainen. 1997. “TI’s new ‘C6x DSP screams at 1,600 MIPS,” Microprocessor Report, February 1997.

[22] H. Safizadeh, H. Noori, M. Sedighi, A. Jahanian, N. Zolfaghari, “Efficient host-independent coprocessor architecture for speech coding algorithms”, Digital System Design, pp: 227-230, 2005

[23] J. Kim, H. Kim, S. Choi, Y. You, “The implementation of G.729 speech coder on a 16-bit DSP chip for the CDMA IMT-2000 system”, IEEE Transactions on Consumer Electronics, vol. 45, no. 2, pp: 443-448, May 1999

[24] TI SPRA564B, “G.729/A speech coder: multichannel TMS320C62x implementation”, Technical Reference, Feb-2000

[25] M. Arora, N. Lahane, and A. Prakash, “All assembly implementation of G.729 Annex B speech codec on a fixed point DSP”, ICASSP, pp: 3780-3783, 2002

[26] M. Banerjee, B. A. Vani, G. R. Krishna, “Optimal real time DSP implementation of ITU G.729 speech codec,” IEEE VTC, pp: 3908-3912, Sep. 2004.

[27] N. Sedaghati-Mokhtari, M. N. Bojnordi, and S. M. Fakhraie, "Efficient simulation of VLIW DSP processors", *in press*.

[28] N. Sedaghati-Mokhtari, M. N. Bojnordi, and S. M. Fakhraie, "Simulation of voice processing applications through VLIW DSP architectures," *in press*.