DRESPA: An Integrated System for Reconfigurable High-speed Signal Processing Applications

Ramadass Narayanadass, Natarajan Somasundaram and J. Raja Paul Perinbam
ramadassn@annauniv.edu boltrani@yahoo.co.uk jrpp@annauniv.edu

Department of Electronics and Communication Engineering
College of Engineering, Anna University, Chennai – 600 025, INDIA.

Summary
This paper describes DRESPA (Dynamically Reconfigurable Embedded Signal Processing Architecture) developed for the class of real-time high-speed signal processing applications. DRESPA is a coarse-grained, multi-programmable and dynamically reconfigurable architecture. The architecture consists of arithmetic operation-level configurable modules interconnected through multiple data buses that can be logically configured to form one or more computation pipelines before a specific application is initiated, and remains unchanged till the completion of the application. On-chip integration of reconfigurable logic reduces the memory access cost and the reconfiguration cost. It neither requires an external processor to configure it, nor does it consume context change-over time. The suitability of DRESPA for the target application domain is evaluated with real-time signal processing applications such as video processing, image processing and speech processing. The results show that there is a performance improvement in terms of speedup in comparison with other systems.

Key words:
Reconfigurable Computing, Embedded System, Digital Signal Processing, High Speed Architecture.

1. Introduction

Reconfigurable systems combine a reconfigurable hardware processing unit with a software-programmable processor. The main goal is to take advantage of the capabilities of both resources. While the processor takes care of all sequential computation, the reconfigurable hardware takes care of specialized vector operations. With such an integrated system architecture, specific properties of applications, such as parallelism, regularity of computation, and data granularity can be exploited by creating custom operators, pipelines, and interconnection pathways.

Real-time digital signal processing applications [1][2] are characterized by a large volume of real-time data input at variable rates and repetitive arithmetic operations on these data. Solutions for such computational problems are mainly driven by the demanded speed and the budget available. The high speed computational requirement usually necessitates hardware based solutions, such as using Application Specific Integrated Circuit (ASIC) [3] or Field Programmable devices [4]. The ASIC will certainly fulfill the speed criteria, but if a slight change in the algorithm occurs, the same ASIC cannot be used. Also, it involves high development cost and development time, besides requiring most of the time for testing and debugging. These drawbacks tend to inhibit the rapid and economical implementation of such systems.

On the other hand, when the speed is not critical, a processor (a software system) can be used. The advantage of this solution is that a change in the algorithm can easily be overcome with a change in the software and therefore be used again. It is clear that processor has more flexibility than the ASIC, but is slower. Beside the speed and the flexibility, the used area is also a consideration. The size of the processor is huge because of the presence of generalized control unit. But ASIC's are comparatively small, since they are application specific and therefore do not need large controlling overhead.

To handle the conflicting requirements of being a flexible architecture and implement some application-specific algorithms, a dynamically reconfigurable embedded architecture is proposed in this paper. The proposed architecture consists of arithmetic operation-level configurable modules interconnected through multiple data buses that can be logically configured to form one or more computation pipelines before a specific application is initiated and remains unchanged till the completion of the application. This architecture is targeted at high-throughput and real-time signal processing applications.

The organization of the paper is as follows: Section II introduces the system model for DRESPA and the proposed architecture is described in detail in Section III. Section IV illustrates the performance evaluation for various digital signal processing applications. The paper concludes finally in Section V.

Manuscript received August 5, 2007.
Manuscript revised August 20, 2007.

2. DRESPA Design Model

The set of criteria that are frequently used to characterize the design of a reconfigurable computing system are granularity, depth of programmability, reconfigurability, type of interface and computation model [5].

Many researchers have proposed other models of reconfigurable systems targeting different applications. Sonic [6] and PADDI [7] are some of the coarse-grain reconfigurable computing systems. Reconfigurable systems with fine-grain granularity include Splash [8], Chimaera [9], Garp [10] and DPGA [11]. Among these, DPGA and GARP are not based on FPGAs. RAW [12] is a mixed-grain reconfigurable system. As evident, all the above systems vary greatly. Since the existing computing systems are not suitable for the target class of real-time applications, a new architecture has to be designed that is capable of handling the target real-time data. The architecture should have a system structure that permits flow of a large volume of real-time data smoothly without any blockage. Since the volume of data is large, it should process data in an on-line fashion; it has to store data in a memory. The major timing constraints of any embedded system come from the periodicity of the input data. All the required tasks for a set of input data must finish before the next set of data comes in. For such a timing requirement, the statistical or probabilistic behavior of the system is undesirable. If there are many statistical factors in the architecture, it is very difficult to ensure the timing correctness of the system, and it is also difficult to trace the timing errors when they ever occur.

In order to flow data freely, the system needs to have some form of pipeline processing. In pipelining multiple instructions are allowed to overlap in execution. Pipeline processing does not involve any statistical or probabilistic behavior; operations are applied on data at each pipeline stage, regularly at each clock cycle. Each pipeline stage makes some contribution to the instruction and can operate in parallel with other stages. Pipelining does not decrease the time to execute individual instructions, but it will execute more instructions per time unit, resulting in speed up.

At one of the system-level pipeline stages, vector operations are performed on the input data in an execution unit. It is not reasonable to have a pipeline for each computation of the application, since it requires an unreasonable amount of hardware. Computational resources must be shared among all the computational aspects of the application to make the amount of hardware reasonable. No matter how complicated a vector operation is, they can be decomposed into arithmetic-operation-level vector operations, such as arithmetic-logic, shift, and multiply operations. Therefore, in the new architecture, the arithmetic-operation-level functional modules are the computational resources to be shared and to be interconnected to form one or more pipelines for the required computations.

The interconnections of the computational resources need to be changed dynamically for different vector operations. A cross-bar switch could provide all the possible interconnections of the computational resources, but would be very expensive. The detailed analysis of the vector computations required by the application can determine the set of computational resources (number and kind) and their possible interconnections. All the interconnections of computational resources should form pipelines to support the pipelined vector operations; each computational resource also needs to be pipelined so that the interconnected resources can be pipelined. Given a set of computational resources, the design of a pipeline for a vector operation determines the processing time, which can be computed from the total number of pipeline stages from input to output. Thus the dynamically reconfigurable computation model addresses the issues of timings as well as those of resource sharing. To cater to the requirements of the real-time high-speed signal processing applications, the embedded architecture should be composed of arithmetic-operation-level configurable modules, that is, granularity should be coarse-grained granularity. The architecture designed in this work puts together, in a cohesive structure, the main prominent features of previous reconfigurable systems, that is, coarse-grain granularity, multiple programmability and dynamic reconfiguration.

The general system architecture, shown in Figure 1, comprises an embedded processor, a reconfigurable unit, a high bandwidth memory unit and input/output (I/O) unit, all implemented as a single chip.

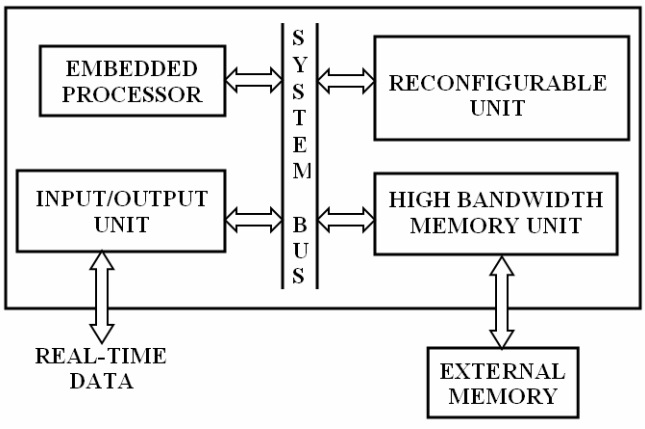

Figure 1. General System Architecture.

Given the nature of target applications, the reconfigurable component is organized as an array or reconfigurable cells (RCs). The RCs are coarse-grained structures. The embedded processor is an autonomous processing unit. It performs scalar operations and controls the operations of RC array. The reconfigurable unit performs vector operations under the control of the embedded processor. A specialized memory unit handles data transfers between external memory and the RC array, and stores input, intermediate and output data. Also, there is a separate memory for storing context data. A dedicated I/O unit handles real-time data input and output.

The performance model for the proposed architecture is based on Amdahl's Law [13]. The performance model gives the theoretical performance achieved by vector operations and pipeline processing, upon which the new architecture is based. They show the limits of the performance gain with vector and pipeline operations.

Let N be the number of processors, s be the time spent by a serial processor on a sequential portion of a program, and p be the time spent by a serial processor on a portion of a program that can be executed in parallel. Then Amdahl's Law says that the speedup S obtained by executing the program with N processors is given by

$$\mathbf{S} = \frac{\mathbf{1}}{\mathbf{s} + \mathbf{p/N}}$$

where s + p = 1. Hence,

$$\mathbf{S} = \frac{\mathbf{1}}{\mathbf{s} + \mathbf{(1-s)/N}}$$

Assume that a task T consists of two portions, T_s and T_v, where T_s is the sequential portion of the task that can be performed only by sequential operations, and T_v the vector portion that can be performed by vector operations. Let r_s and r_v be the fractions of the task corresponding to T_s and T_v, respectively, where $\mathbf{r_s + r_v = 1}$.

Let t_s be the sequential execution time of T, and t_v be the execution time of T when T_v is performed by vector operations, respectively. This is illustrated in Figure 2.

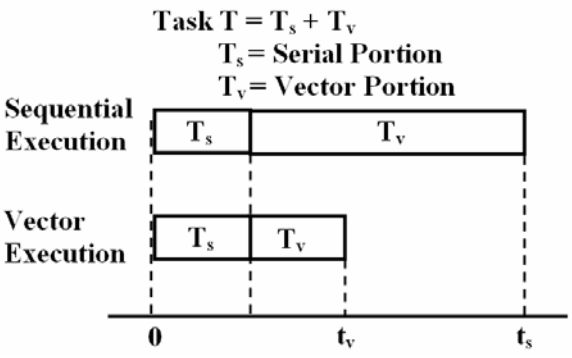

Figure 2. Sequential versus Vector Execution.

Let A be the acceleration factor defined as the ratio of vector operations to sequential operations for T_v. Then, the total speedup S obtained by vector operations for T is represented by

$$\mathbf{S} = \frac{\mathbf{t_s}}{\mathbf{t_v}} = \frac{\mathbf{1}}{\mathbf{r_s + r_v / A}}.$$

Given S and r_s, A is calculated as

$$\mathbf{A} = \frac{\mathbf{1 - r_s}}{\mathbf{1/S - r_s}}$$

For example, if S = 10 and r_s = 0.05, then A = 19. It means that if the required speedup is 10 for an application with the sequential portion of 5%, then a system must be designed that executes the vector operations 19 times faster than the corresponding sequential operations. Since the feasible value of A must be positive, the following condition must be satisfied:

$$\mathbf{r_s . S < 1}$$

That is, the product of the speedup and the sequential portion must be less than 1. This is the theoretical limit on performance gain achieved by vector operations

3. DRESPA System Components

The organization of proposed DRESPA reconfigurable computing system is shown in Figure 3. It is composed of an array of reconfigurable cells (RC array) with context memory, an embedded processor, data buffer and I/O unit. The RC array with its context memory corresponds to reconfigurable SIMD array controlled by an embedded processor, and the high bandwidth memory unit is implemented through data buffer and memory controller. Typically, the embedded processor executes sequential tasks of the application, while the reconfigurable unit (RC array) is dedicated to the exploitation of parallelism available in an application.

These system components are interconnected via four system buses consisting of two I/O buses and two memory buses. Two I/O buses are used to transfer data between Data Buffer and I/O unit; the two memory buses are used to transfer data between Data Buffer and reconfigurable unit. These four system buses allow the system to perform I/O operations and vector operations at the same time. Each bus can be used for any type of data transfer operation. In one of the typical system activities, the following data transfer operations might be performed in parallel:

(1) I/O unit to data buffer: real-time data input.
(2) Data buffer to I/O unit: result data output.
(3) Data Buffer to Reconfigurable unit: Input for vector operations.
(4) Reconfigurable unit to data buffer: Output of vector operations.

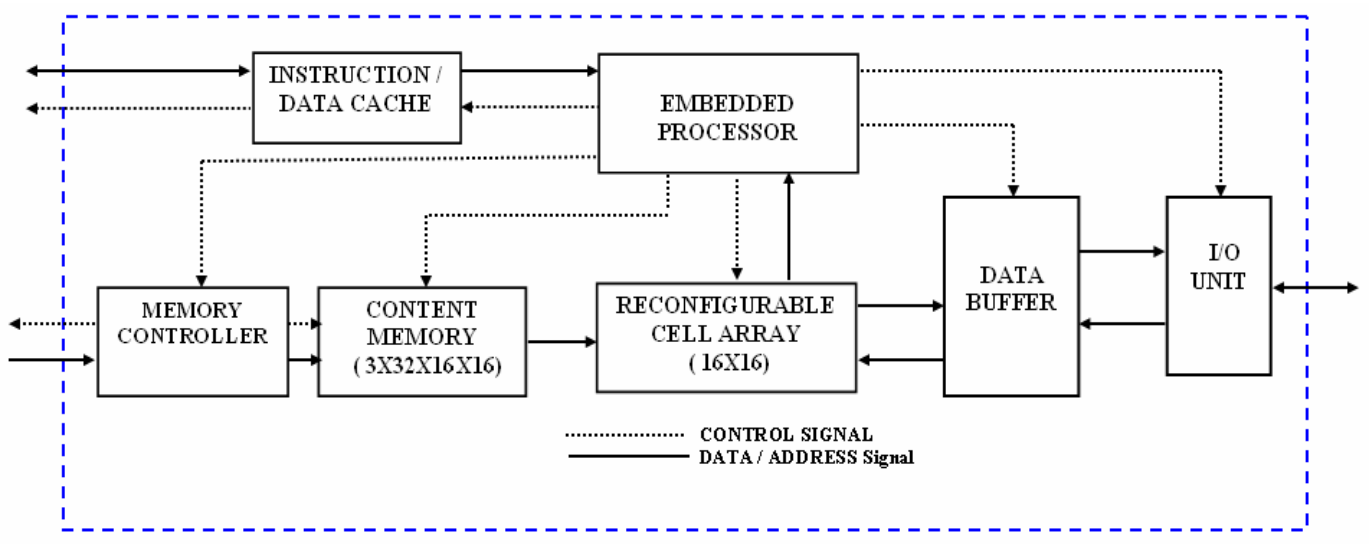

Figure 3. Block Diagram of DRESPA Implementation.

The capability of these four parallel data transfer operations enables the system to operate globally in pipeline. The system-wide pipeline consists of five stages: the first stage for real-time data input, the second for reconfigurable unit data input, the third for vector operations, the fourth for reconfigurable unit data output, the fifth stage for result data output. The time period of this system-wide pipeline is upper-bounded by the cycle time of periodic real-time data, which is the maximum time allowed for data transfer operations and vector operations at the pipeline stages.

The DRESPA system consists of five main components as, embedded processor, reconfigurable cell array, context memory, data buffer and Input/ Output Unit.

3.1 Embedded Processor

The controlling component of DRESPA is a 16-bit embedded processor with a 4-stage scalar pipeline, as instruction fetch, decode, execute and memory operation. It has sixteen 16-bit registers and a 16-bit ALU. It has an instruction and data cache memory which minimizes the access to external main memory. The embedded processor performs scalar operations and controls the other system components through special instructions added to its architecture. It also initiates all data transfers to or from the Data Buffer and configuration program load for the context memory. It is an autonomous processor that executes programs stored in the program memory. It reads machine instructions from the program memory and executes them. Data referenced by an instruction are also fetched from the program memory and the results of the instruction execution are stored back to it. The Controller-Datapath diagram of embedded processor is shown in Figure 4.

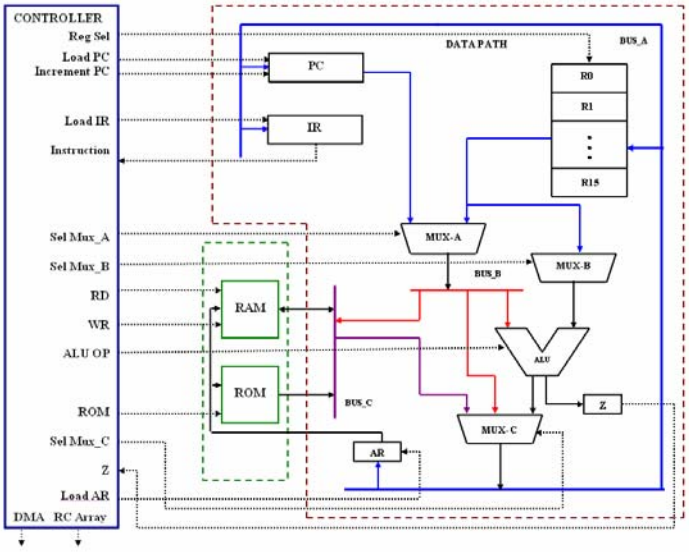

Figure 4. Embedded Processor.

The embedded processor controls the other components through its control signals. This interface is used to initiate vector operations in the reconfigurable unit. In a typical case, the embedded processor first prepares for a vector operation V_i in the reconfigurable cell RC_i and then initiates the reconfigurable unit using the control signal. After the reconfigurable unit starts performing the vector operation, the embedded processor does its own scalar task S_i. When the reconfigurable unit terminates the vector operation, it is sensed by the embedded processor. The embedded processor prepares for the next vector operation V_{i+1} and repeats the process.

3.2 Reconfigurable Cell Array

The basic programmable component of DRESPA is the Reconfigurable Cell (RC). Each RC has an ALU-multiplier and two registers and is configured through a 32-bit context word. The context words for the RC array are stored in context memory. Each RC is the basic unit of reconfiguration. The RC architecture is shown in Figure 5.

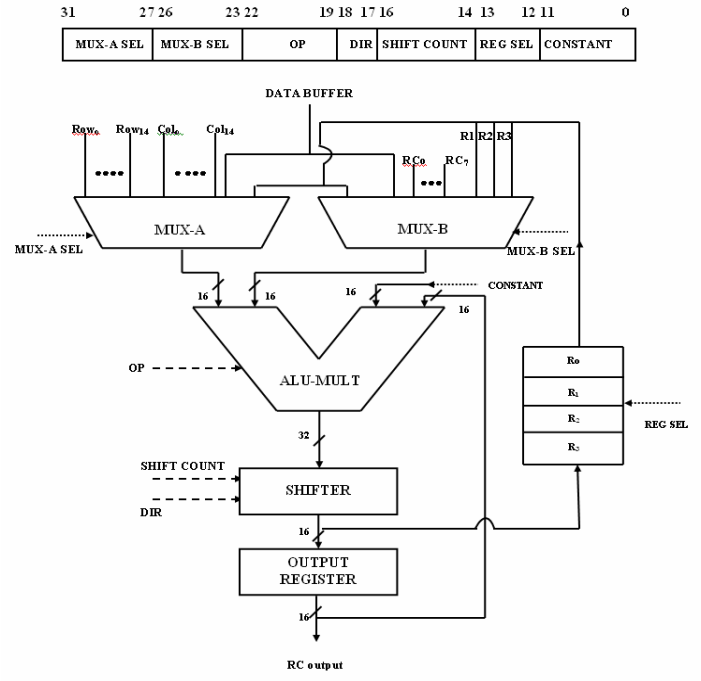

Figure 5. Reconfigurable Cell Architecture.

Each RC comprises of six components: the ALU-multiplier, the shift unit, the input multiplexers, register file with four 16-bit registers, an output register and the context word register. A context word, loaded from context memory and stored in the context register, defines the functionality of the RC. There are 256 RCs, arranged as a 16 x 16 matrix called RC array.

Most multimedia applications required that one of the data input to the multiplier be less than or at most equal to 12 bits. So, the ALU-Multiplier unit includes a 16 x 12 multiplier and a 32-bit ALU. It has four input ports. Two 16-bit ports receive data from the input multiplexers, one 16-bit port takes data from output register and a 12-bit port takes a constant from context word. The shift unit output is also 16-bits wide. There are a total of sixteen ALU functions. The two input multiplexers select one of several inputs for the ALU, based on control bits from the context word in the context register. In addition to arithmetic and logic functions, multiply-accumulate operations can also be performed. The multiplier architecture is carry-save array multiplier. This is the component that has the longest delay in the RC Unit.

Multiplexer MUX_A selects one input from fifteen other RCs in the same row, from fifteen other RCs in the same column, from the Data Buffer, or from internal register. The register file is composed of four 16-bit registers. Multiplexer MUX_B selects one input from eight RCs in the neighborhood, from the Data Buffer, or from internal registers. The feedback register allows reuse of previous operands.

There is a 32-bit register containing the context word for configuring each RC, called as Context Register. It is a part of each RC, whereas the Context Memory is separate from the RC Array. The fields MUX_A and MUX_B specify control bits for the input multiplexers of the RC. The field OP specifies the function for the ALU-Multiplier unit. The fields DIR and SHIFTCOUNT specify the direction and number of shifts, respectively. The field REGSEL determines the register in which the result of an operation is to be stored. The field named CONSTANT is used to supply immediate operands to the ALU-Multiplier unit in each RC. This is useful for operations that involve constants (such as multiplication by a constant over several computations) in which case this operand can be provided through the context word. At the global level, there are buses that run across rows and columns. These buses provide data from any one cell to other adjacent cell.

3.3 Context Memory

The Context Memory provides context words to the RC Array. These context words configure the RC and are also the basis for programming the interconnection network. The context word from the Context Memory is loaded into the Context Register in each RC. It is of size 3 x 32 x 16 x 16 bits or 3 KBytes. Each context word is of size 4 Bytes.

The Context Memory is organized into three blocks which store the contexts for row-wise operation, column-wise operation of the RC Array and additional context respectively. Each block has sixteen sets, with each set corresponding to a row/column of the RC Array. Each set can store thirty two context words. Thus, three contexts can be stored for a single RC. The major focus of this architecture is on regular and data-parallel applications. Based on this idea of parallelism, each context word is broadcast to a row or column of RCs. Thus, all sixteen RCs in a row or column share the same context word, and perform the same operations. They can also have individual contexts. The Context Memory can be updated concurrently with RC array execution. This process is called as dynamic (run-time) reloading of the contexts. Dynamic reconfiguration allows reduction of

effective reconfiguration time to zero.

3.4 Data Buffer

The data buffer is an internal data memory, which is similar to a data cache. It is organized into two sets. Each set has 512 rows of 16 bits each; therefore the data buffer has 512 x 4 Bytes. The memory bus operates in 16-bit mode. This two set organization makes memory accesses transparent to the RC array, by overlapping of computation with data load and store, alternately using the two sets. One of the two sets provides computation data to the RC array and also stores processed data back from the RC Array, while the other set sends processed data to the external interface through the I/O unit and receives data for the next round of computations. These operations proceed concurrently, thus preventing the latency of data I/O from adversely affecting system performance. DRESPA performance benefits tremendously from this data buffer. A dedicated data buffer has been missing in most of the contemporary reconfigurable systems, with consequent degradation of performance.

3.5 Input / Output Unit

The input/output unit depends heavily on the I/O requirements. Input unit takes real-time data at a certain interval and transfers them to the data buffer via one of the I/O buses. Output unit sends processed data to external world. It reads from data buffer through other I/O bus.

3.6 VLSI Implementation

The simulation model of DRESPA is implemented as a HDL model using Verilog HDL. The individual components have been simulated at behavioral level and functional verification is done in Windows platform using Modelsim SE/EE PLUS release 5.4e from Mentor Graphics. The architecture is synthesized into custom ASIC using Leonardo Spectrum release 2002e, for 0.12 μ, 2.5 V, CMOS technology. Table 1 lists the computation time of each of the components as per timing analysis. The RC computational part is the critical path for the entire architecture. It can be noticed that the DRESPA will perform at a clock rate of 75 MHz (or clock period of 13 nS).

4. Performance Evaluation

To evaluate the performance of DRESPA, several signal processing applications were implemented on HDL simulation model of DRESPA. DRESPA achieves a performance improvement and shows a speedup range from 2.27 to 21.00 for different applications. The results

Table 1. Timing Report

Computational Unit	Computation Time (nS)
Embedded Processor	10.11
Reconfigurable Cell (total)	12.92
RC : MUX_A	1.04
RC : 4 to 1 MUX	0.30
RC : 16 x 12 MULTIPLIER	6.40
RC : ALU	2.21
RC : 2 to 1 MUX	0.27
RC : SHIFTER	2.70

show that dynamic reconfiguration helps improve the performance, especially when high-speed applications are executed on reconfigurable structures. The performance speedup obtained for the tested signal processing applications are summarized in Table 2.

Table 2. Performance Report

Application	Speedup
Second order IIR Filter	3.00
MPEG Motion Estimation	16.06
2-D Discrete Cosine Transform	12.30
CELP Algebraic Codebook Search	21.00
AMDF Pitch Estimation	2.27

5. Conclusion

In this paper, the existing major embedded architectures used for system design were reviewed and the proposed architecture, DRESPA was described in detail. Many of these technologies have reached relative maturity. Based on the applications in this work, it appears that the number of contexts does not need to be large to achieve good performance improvement with a Reconfigurable Unit. In these applications, more than one context was used for each application and a considerable speedup was obtained. The question of how many contexts is an optimal number is still unanswered. In case an application used more than two, a configuration allocation algorithm implemented in the compiler could be used to reduce the number of context reconfigurations.

There is still a big gap between the hardware and the software. To close this gap, further investigation is necessary in the area of compilers for reconfigurable embedded systems. It is also desirable to study the behavior of the architecture in presence of operating system. As a final note, investigating the previously mentioned topics will lead to the development of a high performance reconfigurable system. After a complete study of the interactions between architecture, compiler, and operating systems for reconfigurable systems, one would be able to determine the best track to follow in the reconfigurable world.

References

[1] William H. Smith, "Technical Challenges for Designing Personal Digital Assistants", Design Automation for Embedded Systems, 4(1) Jan 1999, pp23-40.

[2] Chu Yu, Hwai Hu, Chen-Yen Lin, "Design and Implementation of an ASIC for 1.6 KBPS Speech Synthesis", in IEEE Transactions on Consumer Electronics, Vol. 49, Issue 3, August 2003, pp.731-736.

[3] M.J.Smith, 1997, "Application-Specific Integrated Circuits", Pearson Education Inc., ISBN: 81-7808-007-9.

[4] S.Brown and J.Rose, 1996, "FPGA and CPLD Architectures: a tutorial", IEEE Design and Test of Computers, pp.42-57.

[5] S.Natarajan, N.Ramadass, 2005, "Self-Modifiable Mixed-Signal SoC Architecture for Embedded Applications", in Proc. National Conference on Signals, Systems and Communications (NCSSC 2005).

[6] Haynes S.D., J.Stone and W.Luk, "Video Image Processing with the Sonic Architecture", Computer: Innovative Technology for Computer Professionals, Vol. 33, No.4, IEEE Computer Society, April 2000, pp.50-57.

[7] D.C.Chen and J.M.Rabey, "PADDI" Programmable Arithmetic Devices for Digital Signal Processing", In VLSI Signal Processing IV, pp.240-249, IEEE Press, Nov. 1990.

[8] M.Rencher and B.L.Hutchings, "Automated Target Recognition on SPLASH", Proc. IEEE Symposium on FPGAs for Custom Computing Machines, April 1997.

[9] S.Hauck, M.M.Hosler and J.P.Kao, "The Chimaera Reconfigurable Functional Unit", Proc. of IEEE Symposium on Field- Programmable Custom Computing Machines, April 1997, pp. 87-96.

[10] Callahan.T.J., J.R.Hauser and J. Wawrzyneck. "The Garp Architecture and C Compiler", Computer: Innovative Technology for Computer Professionals, Volume 33, Number 4, IEEE Computer Society, April 2000, pp. 62-69.

[11] E.Tau, D.Chen and A.DeHon, "A First Generation DPGA implementation", FPD-95, Canadian Workshop of Field –Programmable Devices, May 1995.

[12] J.Babb, M.Frank and V.Lee, "The RAW benchmark Suite: Computation structures for general-purpose computing", Proc. of IEEE Symposium on Field- Programmable Custom Computing Machines, April 1997, pp.134-143.

[13] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," Proceedings of the AFIPS Spring Joint Computer Conference, 1967, pp.483-485.

Ramadass Narayanadass was born in Chennai, Tamilnadu state, India in 1975. He received the B.E. Degree in Electrical and Electronics Engineering from Madras University in 1997, and the M.E. Degree in Applied Electronics from Anna University in 2001. Since 2005, he has been in the Faculty of Information and Communication Engineering, Anna University, where he is currently a lecturer with the Department of Electronics and Communication Engineering.

He is currently working towards the Ph.D. Degree in the Department of Electronics and Communication Engineering at Anna University. His current research interests are Embedded Systems, VLSI Design, Reconfigurable Computing and Embedded Speech Coding.

Natarajan Somasundaram was born in Coimbatore, Tamilnadu state, India in 1980. He received the B.E. Degree in Electronics and Communication Engineering from Barathiyar University in 2002, and the M.E. Degree in Embedded System Technologies from Anna University in 2005.

He was a Junior Scientist under Dr.A.P.J.Abdul Kalam in College of Engineering, Anna University, Chennai. He received a gold medal in his M.E. course. He is selected for Senior Research Fellowship by Council for Scientific and Industrial Research (CSIR), India. He is currently a scientist and working in the Department of Electronics and Communication Engineering at College of Engineering, Anna University, Chennai. His research interests include Mixed-Signal VLSI Design, Image Compression, Reconfigurable Embedded Computing, and related VLSI circuits design.

Raja Paul Perinbam was born in Tamilnadu state, India in 1948. He received the B.E. Degree in Electrical and Electronics Engineering from Madras University in 1970, M.Sc.(Engg) Degree in Applied Electronics from Anna University in 1973, and the Ph.D. Degree from IIT Madras in 1984. Since 1975, he has been in the Faculty of Information and Communication Engineering, Anna University, where he is currently a Professor with the Department of Electronics and Communication Engineering.

He has been a R&D consultant for various UPS based companies. His research interests are Embedded Systems, VLSI Design, Power Electronics and Signal Integrity.