computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

ON CHIP MEMORY REDUCTION TECHNIQUE FOR DATA DOMINATED EMBEDDED SYSTEMS

Srilatha Chepure [1], Guru Rao C.V. [2], Prabhu G. Benakop [3]

[1] Department of ECE, ASTRA, Hyderabad, India-500008
[2] Department of CSE & Principal, KITS, Warangal, India-506015
[3] Department of ECE & Principal, ATRI, Hyderabad, India-500039
E-mail:deepuaurora@yahoo.com, guru_cv_rao@hotmail.com, pgbenakop@rediffmail.com

Abstract: *This paper proposes an approach for optimization of on-chip memory size in data dominated embedded systems. Large amount of array processing is being involved in this category. In order to produce a cost effective system, efficient designing of memory module is quite critical. The memory module configuration being selected by the designer should be well suitable for the application. In this regard, this paper presents a methodology for effective optimization of on-chip memory. For sensitive applications involving large array processing, the entire processing has to be done using embedded modules. While using such modules, care should be taken to meet optimized profile for the design metrics. With help of loop transformation technique, relatively a good amount of memory size requirement is reduced for the arrays. This approach results in a very close memory estimate and an effective optimization. This methodology can be further extended to meet the high level memory optimization applications based on cache characteristics. Speech processing front end mechanism is implemented and shows that this approach gives up to an achievement 61.3% reduction of overall system memory requirement over the estimation approach. Results are provided in terms of comparison of the two approaches of memory estimation and optimization with respect to both of the program and data segments.*

Keywords: *Embedded systems, memory, estimation, and optimization.*

1. INTRODUCTION

In today's embedded systems, memory represents a major bottleneck [1] in terms of cost, performance, and power. Optimal designing of memory space is very crucial in embedded system designing. Also, a large amount of array processing is being involved in current day embedded applications. Hence, it is very critical to come out with methodologies for memory size estimation and optimization. In embedded applications involving large amounts of data processing i.e. Data dominated embedded systems, much power consumption is because of the global communications and memory hit/miss rates. Thus it is important to estimate the memory requirements for the data structures and code segments for that particular application. Memory requirement is defined as the number of locations needed to satisfy the storage requirements of a system. It is very important to effectively predict the system's memory requirements without synthesizing, in order to obtain a high profile end product, as it results in a reduced design time. In this paper, we present an optimization strategy for efficient on-chip memory requirement. Here memory optimizing transformations are employed to reduce the memory size and number of accesses. This aim at reusing of memory space, thus giving a fast estimate of memory size. Though addressing becomes complex, it is preferable to allow sharing among arrays which aids in optimizing the memory size.

Consider: int a [xyz];
int b [xyz];

This involves two arrays in sequential order. As said above, if sharing is allowed between arrays, the memory size reduces as follows:

```
Struct share
{
    int a;
    int b;
}
struct share shared_array[xyz];
```

After allowing sharing between arrays, it involves only one structured array. Here the array sharing removes the conflicts between a and b there by improving spatial locality.

The paper is organized as follows: Section 2 briefly reviews some previous work done in the area

of memory estimation and optimization. The proposed methodology is described in Section 3. Section 4 gives a brief description about an exemplary data dominated embedded system along with its memory requirements, while its experiment set up is explained in section 5. Section 6 and 7 shows the results of the task implemented and its conclusion.

2. RELATED WORK

Embedded applications have a built in hierarchy. An application is composed of several modules, where each module consists of one or more code and data segments. [2,3] employed optimization by placing of frequently accessed data variables in on-chip SPRAM and placing less frequently accessed data variables in off chip RAM. Partitioning data arrays that are accessed simultaneously in the same processor cycle into different on-chip memory banks [4, 5] forms a good optimization for array dominated systems. [6, 7] showed that swapping critical code and data segments from off-chip memory to on-chip memory before the execution of the appropriate code segment aids in efficient optimization. Except for the swapping technique, which works on both code and data, all the other techniques concentrate only on data. Managing data is very important because most of the embedded applications are data dominated [8]. Stochastic search methods using genetic algorithms [9] were heuristic. Storage allocation methodology [10] employed compliers for estimation. Our approach optimizes memory module, while [11] dealt only with memory allocation process. For general purpose systems whose area of application is wide, the dynamic memory allocation is supported by custom managers [12]. Also, [13, 14] showed memory optimizations and techniques to reduce memory footprint along with power consumption and performance factors on static data for embedded systems. Array based data flow preprocessing considers program size as well as data size [15] is applicable only for partially fixed execution ordering. In [16], the design metric constraints were area and number of cycles, while the proposed methodology also considers power consumption. Live variable analysis along with integer point counting method [17] is not applicable for large multi-dimensional loop nest as it needs complex computations. [18] Is based on analysis of memory size behavior taking into account that signals with non-overlapping lifetimes share same memory locations. Memory system design for video processors [19] had constraints on area, cycle time. [20] proposed data memory size and number of cycles as design metrics. Memory allocation problem [21] was solved by meeting optimum cost but efficient memory access modes were not exploited. To reduce the power consumption, number of off-chip accesses as well as size of storage during memory optimization, loop transformation reordering is presented in this methodology which is much more beneficial. This proposed methodology is validated by performing experimentation on a data dominated communication module. Our approach even works for multimedia applications involving large array processing.

3. APPROACH

The output of our approach is an optimized estimate of the memory size. This paper describes a procedure for memory optimization for low power embedded systems. Here the system consists of a register file, a data cache and an instruction cache on-chip, and a large memory off-chip. The first step of the procedure is application of memory optimizing transformations to reduce the memory size and number of accesses. In involves the application of loop transformations to reduce power in data dominated applications.

Loop transformation aims at regularity and locality of reference. It basically involves the following:

a) Loop reordering
b) Loop fission
c) Loop interchange
d) Loop fusion

Loop reordering allows arrays to share memory space, thereby reducing the size of the on chip memory. Loop interchange helps to reduce the number of memory reads. The number of memory accesses and the size of storage significantly reduce. However, each transformation has its own special legality test based on the direction vectors and on the nature of loop bound expressions.

a) Loop reordering

Here the loops which employ arrays that are not alive in the rest of the code are placed at the top such that off chip memory size is reduced. Thus saved memory can be used to accommodate other arrays.

```
Consider:   Loop1: For (i= 0; i<N; i++)
                       p[i]=q(b[i])
            Loop 2: For (i= 0; i<N; i++)
                       r[i]=f(s[i])
            Loop 3: For (i= 0; i<N; i++)
                       a[i]=f(b[i],c[i],d[i])
            Loop 4: For (i= 0; i<N; i++)
                       t[i]=f(u[i])
After Loop reordering:
            Loop 3: For (i= 0; i<N; i++)
                       a[i]=f(b[i],c[i],d[i])
            Loop1: For (i= 0; i<N; i++)
```

```
        p[i]=q(b[i])
Loop 2: For (i= 0; i<N; i++)
        r[i]=f(s[i])
Loop 4: For (i= 0; i<N; i++)
        t[i]=f(u[i])
```

b) Loop fission

For the loops do not have any data dependencies or which involves different access patterns, loop fission is implemented.

Consider:

```
For (i= 0; i<N; i++)
    For (j= 0; i<N; j++)
    p[i,j]=f(p[i,j-1])
    q[i+1,j]=f(q[i-1,j])
```

After loop fission

```
For (i= 0; i<N; i++)
For (j= 0; i<N; j++)
p[i,j]=f(p[i,j-1])
For (i= 0; i<N; i++)
For (j= 0; i<N; j++)
q[i+1,j]=f(q[i-1,j])
```

c) Loop interchange

It aids in reducing the memory accesses. Also, it increases the variable usage by which they can be easily stored in registers instead of storing in memory module. Thus, the amount of on chip memory reduces.

Consider:

```
for (n = 0; n < 100; n = n+1)
for (m = 0; m < 100; m = m+1)
for (l = 0; l < 2000; l = l+1)
        a[i][j] = 4 * a[i][j];
```

After loop interchange

```
for (n = 0; n < 100; n = n+1)
for (l = 0; l < 2000; l = i+1)
for (m = 0; m < 100; m = j+1)
        a[i][j] = 4 * a[i][j];
```

d) Loop fusion

It helps in reducing the number of memory accesses and also the size of off-chip memory. This is done is done if there are data dependencies between the two fusing loops. This is because loop fusion causes an increase in the size of the loop body which in turn causes an increase in the minimum cache size which in turn causes an increase in the energy consumption.

Consider:

```
for (i = 0; i < N; i = i+1)
for (j = 0; j < N; j = j+1)
        a[i][j]= 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j]+ c[i][j];
```

After Loop fusion

```
for (i = 0; i < N; i = i+1)
for (j = 0; j < N; j = j+1)
{
    a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
}
```

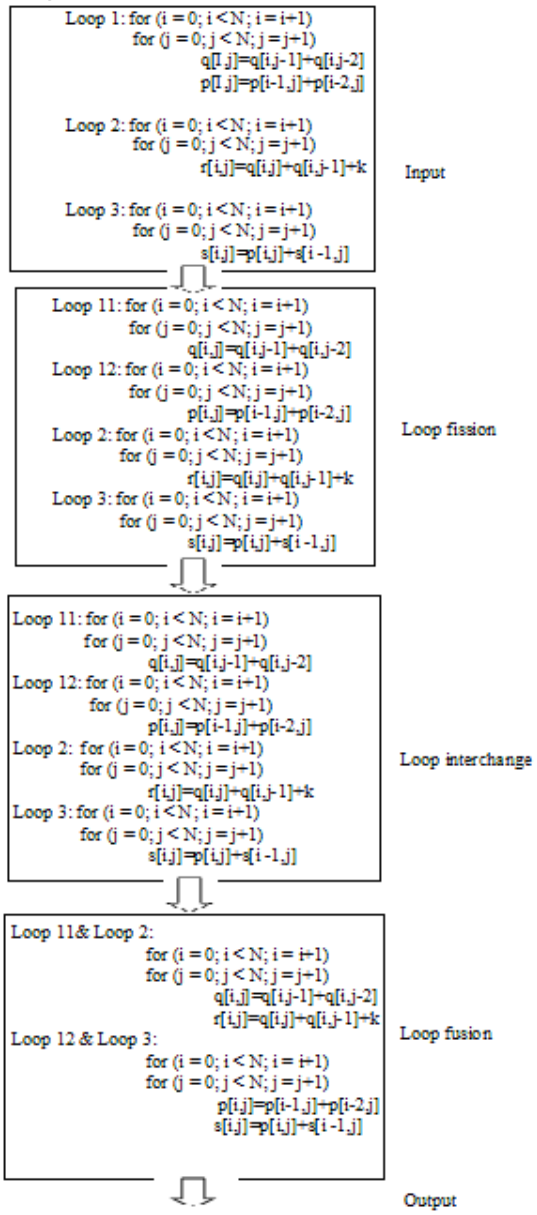

Fig. 1 – Loop transformation flow

Thus the loop transformation technique reduces memory requirements, as each iteration involves fewer references. This in turn improves the cache performance significantly. But splitting of the involved references might reduce the number of dependencies inhabited in the loop.

4. AN EXEMPLARY DATA DOMINATED EMBEDDED SYSTEM: SPEECH RECOGNITION MODULE

Speech recognition is one of the most significant real time embedded application. In this, the entire signal processing front end mechanism has to be done using embedded modules. It basically requires

efficient memory analysis as they are small in size and are battery powered. As a result, memory analysis of such a system is very valuable for system design. Figure shows the speech recognition system block diagram. The decoder's computation is iterative, and each iteration processes a new observation vector from the speech front end. In every iteration, each state in the recognition network executes two steps:

1) Computation of observation probability for the current observation

2) Examination all the incoming tokens and selecting the best one.

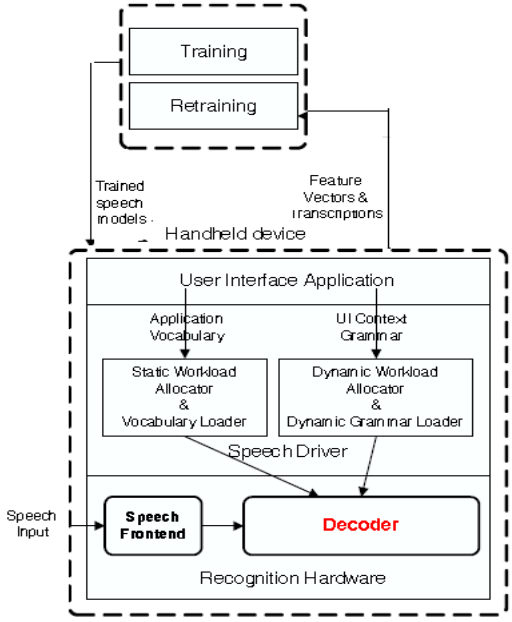

Fig. 2 – Speech recognition block diagram

Markov method is employed for time variants having discrete state spaces. Each of the discrete space state gives out speech perceptions as per its probable distribution. Thus obtained speech perceptions can be either discrete or continuous. They basically represent frames. As the states cannot be observed directly, it is termed as hidden Markov model. The following is the speech recognition algorithm. It consists of two parts. First is the search algorithm and second is the processing part.

Memory bandwidth, a traditional bottleneck in parallelizing speech computation, can be easily overcome by integrating multiple blocks of memory along with required logic on the same chip. This reduces the power consumption as well as memory access latencies.

The system has RAM memory to hold the following:

- Frame length
- Real part of the intermediate FFT radix-2 stages.
- Imaginary part of the intermediate FFT radix-2 stages.
- Mel-filter spaced values
- Quantization tables

Along with the above said, ROM is required to store the following:

- To store Hamming window factors.
- Twiddle factors
- DCT factors

5. EXPERIMENTATION

The methodology employed the Texas Instruments TMS320C6701 processor for the experiments and Texas Instrument's Code Composer Studio (CCS) environment for obtaining the profile data. The program memory consists of a 64K-byte block that is user-configurable as cache or memory-mapped program space while the data memory consists of two 32K-byte blocks of RAM. Code Composer Studio V2.2 [22] is employed to run the applications. Initially the applications are compiled with the CCS2.2 compiler with the default memory placement made by the compiler. The compiled application is loaded and simulated in the simulator to obtain the profile information. [23,24,25]. the main inputs for experiments on the speech recognition module are the access characteristics of the data segment. Also TI's ASIC memory library is used for the memory allocation step. The kernels of the applications are developed in hand optimized assembly code. The profile data is obtained by running the compiled executable in a cycle accurate software simulator. For obtaining conflict data we used a bank of single-access RAM that fits the application data size. The output profile data contain frequency of access for all data sections.. In the due process, the simulation and the estimation based approaches are analyzed with respect to each of the results.

6. RESULTS

Memory trace

The following are the memory estimation and optimization values obtained.

Parameter	Memory Estimation	Optimized memory
Data segment	14 KB	7.1 KB
Program segment	68 KB	43.2 KB
Total module	82 KB	50.3 KB

Memory trace for the implementation is shown in fig. 3. It considers program and data segments on the X axis and required memory size on the corresponding Y axis. A plot of it results in a memory trace which is the estimated size that caters the storage requirements of both program and data segments in accomplishing the task of signal processing front end mechanism.

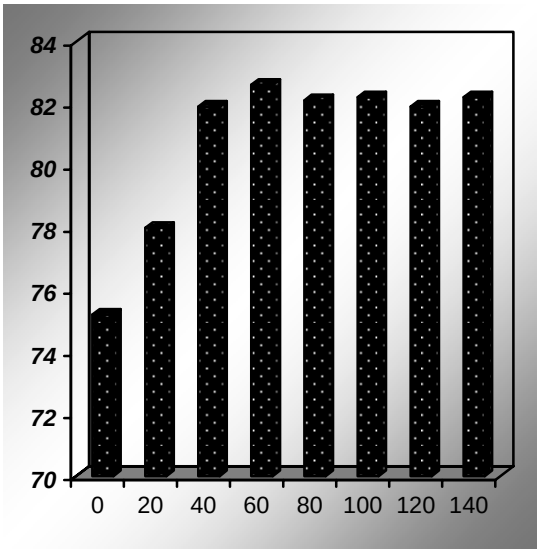

Fig. 3 – Memory estimation for Program and data segments

Memory optimization

Fig 4 shows an effective trace yielding in optimized memory requirement. It considers program and data segments on the X axis and required memory size on the corresponding Y axis. With help of loop transformation techniques this methodology results in a reduction of 31.7 Kbytes.

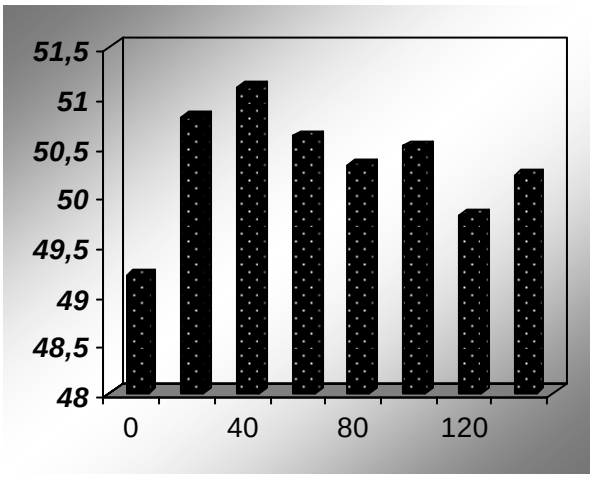

Fig. 4 – Memory optimization for Program and data segments

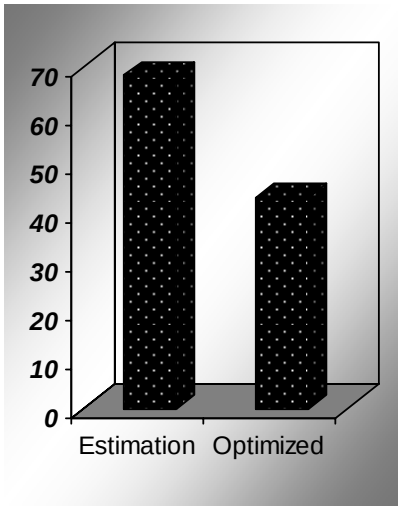

Fig. 5 – Memory estimation and Optimization depiction for program segment

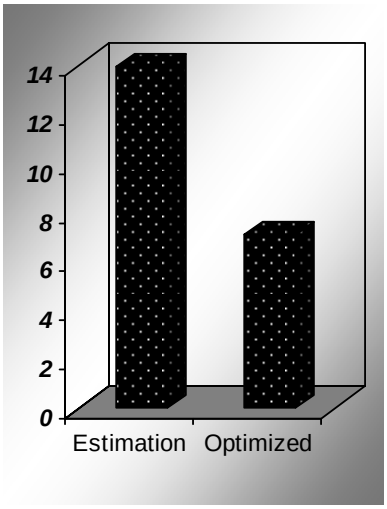

Fig. 6 – Memory estimation and Optimization depiction for data segment

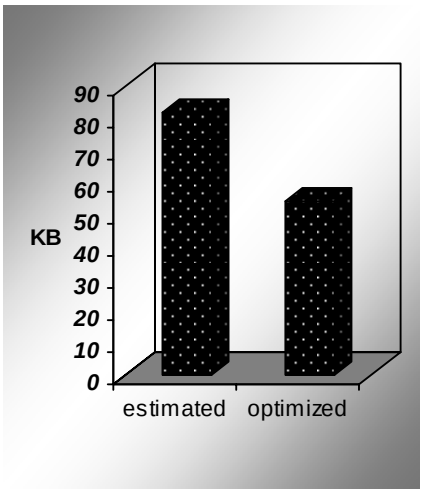

Fig. 7 – Total Memory estimation and Optimization depiction

7. CONCLUSIONS

This paper proposes an optimization strategy for memory module in low power embedded systems. The approach presented efficiently optimizes the memory module, in turn optimizing the design time. Loop transformations are applied to reduce the number of off chip memory accesses and also the on chip memory requirement. Also, the methodology is validated by performing experiments on an embedded speech Recognition module, showing an effective reduction in the memory requirement of the system. In this approach, loop level transformations are applied for memory optimization, which considerably reduces the number of memory accesses. Depending upon the results, even algorithm based optimization can be done with an aim of further reducing the memory size.

ACKNOWLEDGEMENT

This work was partially funded by Aurora's Scientific Technological & Research Academy, India.

REFERENCES

[1] Peter Grun, Nikil Dutt, and Alex Nicolau, Access Pattern Based Memory and Connectivity Architecture Exploration, *ACM Transactions on Embedded Computing Systems*, (2) 1 (2003), pp. 33-73.

[2] O. Avissar, R. Barua, and D. Stewart, Heterogeneous memory management for embedded systems, *in Proceedings of ACM 2nd International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, November 2001.

[3] P. R. Panda, N. D. Dutt, and A. Nicolau, *Memory issues in Embedded Systems-on-chip: Optimizations and Exploration*, Kluwer Academic Publishers, Norwell, Mass., 1998.

[4] M. Ko and S. S. Bhattacharyya, Data partitioning for DSP software synthesis, *in Proceedings of the International Workshop on Software and Compilers for Embedded Processors*, September 2003.

[5] M. A. R. Saghir, P. Chow, and C. G. Lee, Exploiting dual data-memory banks in digital signal processors, *In Proceedings of the 7th Intl Conference Architectural Support for Programming Languages and Operating Systems*, (October 1996), pp. 234-243.

[6] M. Kandemir, J. Ramanujam, and A. Choudhary, Improving cache locality by a combination of loop and data transformations, *IEEE Transactions on Computers*, 1999.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vechi, *Optimization by simulated annealing*, Science, 1983, 220 p.

[8] F. Catthoor, N. D. Dutt, and C. E. Kozyrakis, How to solve the current memory access and data transfer bottlenecks: at the processor architecture or at the compiler level? *In Design, Automation and Test in Europe Conference and Exhibition*, (2000), pp. 426-433.

[9] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, Wiley, 2003.

[10] J. Sjodin and C. Platen, Storage allocation for embedded processors, *in Proceedings of ACM 2nd International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, November 2001.

[11] J. Seo, T. Kim, and P. Panda, Memory allocation and mapping in high-level synthesis: an integrated approach, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, (11) 5 (2003).

[12] RTEMS Research, O.-L. A. RTEMS, Open-source real-time operating system for multiprocessor systems, 2002.

[13] Panda P.R., Catthoor F., Dutt N.D., Danckaert K., Brockmeyer E, Kulkarni C., Data and memory optimizations for embedded systems, *ACM Trans. Des. Automat. Elect. Syst*. (6) 2 (2001), pp. 142-206.

[14] Benini L., De Micheli G., System level power optimization techniques and tools, *in ACM Trans. Des. Automat. Embed. Syst*. 2000.

[15] P. G. Kjeldsberg, F. Catthoor, E. J. Aas, Storage requirement estimation for data intensive applications with partially fixed execution ordering, *Proceedings of 8th International Workshop on Hardware/Software Codesign*, San Diego, (May 3-5, 2000), pp. 56-60.

[16] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, Influence of Compiler Optimizations on System Power, *37th IEEE/ACM Design Automation Conference*, (2000) pp. 304-307.

[17] Y. Zhao, S. Malik, Exact memory size estimation for array computations without loop unrolling, *Proceedings of 36th ACM/IEEE Design Automation Conference*, New Orleans LA, (June 1999), pp. 811-816.

[18] P. Grun, F. Balasa, N. Dutt, Memory size estimation for multimedia applications, *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, Seattle WA, (March 1998), pp. 145-149.

[19] S. Dutta, W. Wolf, and A. Wolfe, A methodology on evaluate memory architecture

design tradeoffs for video signal processors, *IEEE Transactions on Circuits and Systems for Video Technology*, (8) 1 (1998).
[20] P. R. Panda, N. D. Dutt, and A. Nicolau, *Data Cache Sizing for Embedded Processor Applications*, Technical Report ICS-TR-97-31, University of California, Irvine, June 1997.
[21] H. Schmit and D. E. Thomas, Array mapping behavioral synthesis, *ISSS*, 1995.
[22] Texas Instruments, http://focus.ti.com/dsp/docs/. *Code Composer Studio (CCS) IDE*.
[23] TMS 320C6201/6701 Evaluation Module, Technical Reference, Texas Instruments.
[24] TMS320C6000 Code Generation Tools Online Documentation (SPRH014E) 1998-2000 Texas Instruments Incorporated.
[25] The TMS 320C6X Optimizing C Compilers User's Guide (SPRU 187), Texas Instruments.

Ms. Srilatha Chepure *received her Bachelor's Degree in Instrumentation Engineering from Osmania University, Hyderabad, India. She is a Master degree holder in Embedded Systems from Jawaharlal Nehru Technological University, Hyderabad, India. Currently, she is an Assistant Professor at Aurora's Scientific Technological & Research Academy. She has six years of teaching experience at college level. Her area of interest includes embedded systems, Real time systems. She is carrying out her research work in the field of embedded systems under the guidance of Dr. C V Guru Rao, Principal, KITS College, Warangal, India. She has four International paper publication to her credit. She is a life member of Computer Society of India and Instrumentation Society of India.*

Dr. Guru Rao C.V. *received his Bachelor's Degree in Electronics & Communi-cations Engineering from VR Siddhartha Engineering College, Vijayawada, India. He is a double post graduate, with specializations in Electronic Instrumentation and Information Science & Engineering. He received his M.Tech in Electronic Instrumentation from Regional Engineering College, Warangal, India and M.E in Information Science & Engineering from Motilal Nehru Regional Engineering College, Allahabad, India. He is a Doctorate holder in Computer Science & Engineering from Indian Institute of Technology, Kharagpur, India. With 24 years of teaching experience, currently he is the Professor and Head, department of CSE, SR Engineering college, Warangal, Andhra Pradesh, India. He has more than 35 National and International publications to his credit. He is the Chairman, Board of Studies for Computer Science & Engineering and Information Technology, Kakatiya University, Warangal. Also, he is the Editorial Board member for International Journal of Computational Intelligence Research and Application journal. He is a life member of Indian Society for Technical Education, Instrumentation Society of India, and member of Institution of Engineers, Institution of Electronics & Telecommunications Engineers and Institution of Electrical & Electronics Engineers (USA).*

Dr. Prabhu G. Benakop*, Professor in ECE and the Director of ATRI. He has 24 years of teaching experience. His research interest is in the areas of Microprocessors, Computer Networks and VLSI. He has 32 publications in various national and international conferences and journals. Presently 8 research scholars are working under him for their Ph.Ds. He is a Senior member of IEEE and life member of ISTE,ISOI.*