# High Level Synthesis of Multi-precision Data Flow Graphs

Vikas Agrawal
Supercomputer Education and Research Center,
Indian Institute of Science,
Bangalore-12.
vikas@serc.iisc.ernet.in

Anand Pande*
Broadcom India Pvt. Ltd.
Sarafarazi,
87/27, Richmond Road,
Bangalore-25
anand.pande@broadcom.com

Mahesh M. Mehendale
Texas Instruments (I) Ltd.
Golf View Homes,
Murugeshpalaya,
Bangalore-17
m-mehendale@ti.com

## Abstract

*A number of DSP algorithms involve linear transforms employing weighted sum computations, where the weights are fixed at design time. Add-shift implementation of such a computation results in a Data Flow Graph that has multiple precision variables and functional units. We explore the potential of precision sensitive approach for the high level synthesis of such multi-precision DFGs. We focus on fixed latency implementation of these DFGs. We present register allocation, functional unit binding and scheduling algorithms to exploit the multi-precision nature of such DFGs for area efficient implementation. The proposed approach is fairly generic and could be applied to multi-precision DFGs involving any type of functional units. Significant improvements of upto 27% have been obtained over the conventional high-level synthesis approach.*

## I. Introduction:

DSP applications are based on DSP kernels such as filtering (FIR and IIR) and transforms (DCT and FFT etc.). Many of these transforms involve weighted sum computations wherein the weights are fixed at the design time. Add-Shift based hardware implementation of such fixed-coefficient multiplications is preferred over a multiplier based realization for both area and performance efficiency. The number of computations in such implementation can be minimized with techniques like common sub-expression elimination [1-3,10]. The computational DFG (Data Flow Graph) structure depicting such add-shift implementation comprises variables and computations whose precision varies significantly. For example, consider a 32-tap FIR filter with 16-bit coefficients and 12-bit data. The resultant DFG in add-shift implementation would have variables and computations with precision varying from 12 bits at input to as high as 33 bits at the output.

In this paper, we address the area efficient resource shared implementation of such multi-precision DFGs. This problem can be solved as a conventional high-level synthesis problem. However, since the techniques involved in the conventional high-level synthesis ignore the varying precision nature of DFG, the resultant implementation is likely to be sub-optimal. We present a precision sensitive behavioral synthesis technique where we address all the three major components of high level synthesis, viz. register allocation, functional unit binding and scheduling. In each of these steps we exploit the multi-precision nature of the DFG to achieve most area efficient implementation. Since these three components of high-level synthesis are interdependent [7], we present an integrated methodology to take advantage of the coupling between them.

To the best of our knowledge the problem of supporting multi-precision arithmetic hasn't been looked at in the context of high-level synthesis of ASICs (Application Specific Integrated Circuits). Some processor architectures have been proposed to efficiently perform variable precision computations, mostly with the granularity of 8/16/32 bits [8,9]. Multi-precision arithmetic in context of Image and Video Processing applications is dealt with in [4]. The inherent parallelism of such applications is exploited for multi-precision functional unit allocation and binding. Here also, the multi-precision arithmetic is restricted to byte or word boundaries. Restricting to word or byte boundaries is often necessary in processors to optimize memory-cpu transfers. This restriction does not apply to ASICs and the scope of area optimization is significantly enhanced. Moreover, scheduling has not been addressed in [4], considering massive parallelism available in the Image and video

*This work was done when the author was with Texas Instruments India Ltd.

0-7695-0831-6/00 $10.00 © 2000 IEEE

applications. Such parallelism is not present in multiplier-less implementation of weighted sum transforms and hence those techniques can't be directly applied

Although most of our techniques are described in the context of add – shift implementation, they are generic enough and could be applied to multi-precision DFGs involving any type of functional units. The rest of the paper is organized as follows. In the next section we describe the register allocation and functional unit-binding problem in the context of multi-precision arithmetic. A precision sensitive register allocation and functional unit-binding algorithm is also presented. Section III describes scheduling and the integrated HLS (High Level Synthesis) methodology. Results are presented in section IV. These results justify our claims of area efficiency over conventional methods. Conclusions are drawn in section V and areas of future work are outlined.

## II. Register allocation and functional unit binding:

In this section we address the problem of register allocation and functional unit binding in the context of multiple precision DFGs. The objective is to exploit the varying precision of the DFG to attain maximum area efficiency. The register allocation problem has been extensively studied in the literature [6,7,11,12]. The input to the register allocation problem is the variable lifetime graph, which is derived from the given schedule of the DFG. The following example illustrates the improvement that can be achieved using precision sensitive approach for register allocation. Consider the variable lifetime graph of a 4 variable Data Flow Graph as shown in the figure 1. The conventional left-edge allocation algorithm [12] will allocate variables A,C and B,D to two registers Reg1 and Reg2 respectively. This allocation is insensitive to the precision of the variable. If variables A,B,C and D have bit precision of 8,15,16,9 respectively the resultant implementation would require two registers of 16 and 15 bits respectively.

On the other hand a precision sensitive approach for allocation should allocate variable A,D and B,C to two

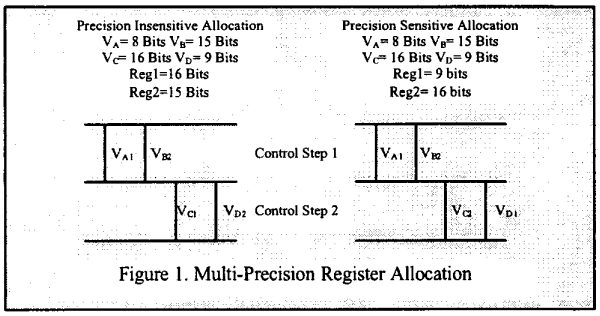



Figure 1. Multi-Precision Register Allocation

precision respectively. We now present a precision sensitive, register allocation algorithm. The pseudo code for this algorithm is shown in figure 2.

### II(A). Precision sensitive register assignment algorithm :

For a given schedule of the DFG, the input to this algorithm is the lifetime graph of variables. The 2-tuple <start(v), end(v)> characterize the lifetime of each variable. Similar to the left-edge algorithm [12], the variables are sorted with <start(v)> of their lifetime as the primary key. But instead of having secondary key as the <end(v)>, it is taken as bit precision of the variables in decreasing order. The algorithm maintains two ordered working lists ("free_unit" and "cur_var") and a regular list ("busy_unit"). The list "free_unit" holds the units ( registers in this case) freed at a control step. The list "busy_unit" holds the list of units currently holding some variable. These two lists get modified as the variables are assigned to units and as the units are freed due to the completion of lifetime of variables in each control step. List "cur_var" holds the variables having their <start(v)> same as the control step being considered.

Algorithm starts from *0th* control step with "free_unit", "busy_unit" being empty lists. Variables having their <start(v)> equal to 0 are added to the list "cur_var". At each control step, the members of the "cur_var" list are assigned to the members of the "free_unit" list in order. These units holding variables are moved to the list "busy_unit".

```
Procedure multi_prec_allocate (var_lifetime_graph)
busy_unit = empty;
free_unit = empty;
for each control step i= 0 to MAX_STEP
    1. cur_var = NULL;
    2. new_freed_units = Those busy units whose end life time
    corresponds to control step i; /*new_freed_units is different
    from free_units*/
    3. Move freed units from busy_unit to free_unit maintaining
    the decreasing bit precision order of list free_unit ;
    4. Add all the variables with <start(v)>== i to cur_var list
    in order of decreasing precision;
    5.Assign each variable from cur_var to the corresponding
    unit in list of free_unit in order and move the assigned units
    from free_unit to busy_unit ;
    6. Assign any leftover variables in cur_var to new units and
    move those units to busy_unit;
end - for
end - procedure multi_prec_allocate
```

Figure 2: Algorithm for Register Allocation and FU Binding

Any unallocated variables in "cur_var" list are allocated to new units and these units are also added to the list "busy_unit". At each control step the any unit being freed from the list of 'busy_unit" are moved to ordered list "free_unit" maintaining the order of decreasing precision. When all the variables in "cur_var" get allocated, the algorithm continues to next control step and performs the same operations. This continues till all control steps are exhausted.

The foundation of the register allocation algorithm is laid on the following facts:

a. The variation in bit precision of the register should be minimum. This ensures optimum utilization of a register.

b. The number of the registers used should be minimum similar to the left edge algorithm.

## II(B). Functional unit binding

The effect of precision sensitive approach for functional unit binding is shown in the figure 3. Consider the DFG shown in figure 3. A precision insensitive approach could result in implementation as shown in figure 3(A), having two adders of 18-bit and 19-bit width. Where as a precision sensitive approach to functional unit binding shall bind the operations as shown in figure 3(B) requiring two adders of 13-bit and 19-bit width respectively. The functional unit binding can be viewed as a special case of register assignment problem where the lifetime of computation is always one control step. Note that this kind of a formulation gives us a flexibility to handle multi-cycle operations as well. Here again we use similar algorithm as register allocation algorithm to bind operations to functional units.The minor changes being:

a. Each computation is considered as a variable having lifetime of one control step.

b. Each type of computation (adder ,shifter) is considered separately.

## III. Scheduling:

Depending upon the application in consideration the scheduling can be resource constrained or time constrained [7]. Most of the DSP applications, being used in real time systems are tightly constrained with time rather than resources. Keeping this in mind, we address time constrained scheduling of multi-precision DFGs. The objective here is to obtain an area efficient implementation of the computational structure represented by the DFG, without increasing its latency.

A resource shared multi-precision multiplier-less DFG implementation comprises adders, shifters, registers, multiplexers and interconnect. While the conventional approach minimizes the number of functional units we aim to minimize the number of bits of the functional units. Qualitatively, this can be understood with the help of following simple example. Consider the DFG shown in the figure 4(A). The function W(operator) refers to the bit width of operator. For example if addition operator "A" requires precision of 10 bits then $W(O_A)=10$. Precision insensitive time constrained scheduling methodology designed to minimize the number of functional units may schedule the DFG requiring two adder units of Width 16 bits and 17 bits respectively. A precision sensitive scheduling technique would require two adders of width 12 and 17 bits respectively (figure 4(C)). The relative effect of bit-width on the area of the functional and storage units is discussed next, which further enhances our approach of area minimization. Since scheduling, allocation and binding are interdependent [7], we address the problem of scheduling allocation and binding in an integrated manner. We use an iterative improvement methodology, based on the implementation-cost minimization. Register allocation and functional unit binding are actually performed for each of the feasible intermediate schedules to obtain the implementation cost.

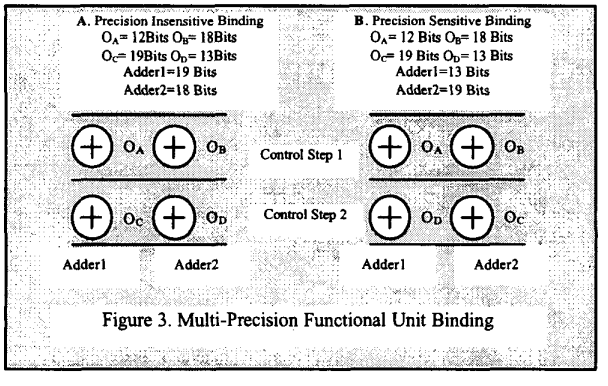


Figure 3. Multi-Precision Functional Unit Binding

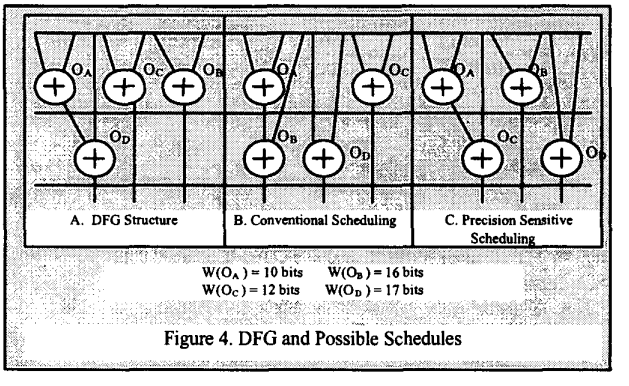


Figure 4. DFG and Possible Schedules

The computational complexity of both register allocation and functional unit binding proposed by us in section II, is polynomial time. Hence, we can afford to do these computations for each of the schedules, rather than relying on some estimation mechanism. This modification contributes to the quality of the implementation and significant gain is achieved without being computationally expensive.

## III(A). Cost function

In this subsection we show the relation between number of bits of functional units and their area. Since we aim to reflect the implementation area in the cost function formulation, we propose to associate different weights per bit to the each of the resources (adder, shifter and storage). The relative weights are dependent upon the technology used and the type(architecture) of functional units employed. Now, we illustrate the computation of the relative weights associated with each of the resources in our approach. To keep our technique fairly generic we have used a generic block synthesis tool Synopsys MODULE COMPILER to synthesize the adders, shifters and registers for a number of bit widths. We have used the tool in area optimization mode with the Texas Instruments 0.15 micron ASIC library. The area indicated is in terms of equivalent NAND gates The objective of performing synthesis of each of the resources is to establish the validity of our cost function quantitatively. It is however, not necessary to perform the whole synthesis and approximate weights can be assigned the bits of each resource and it's architecture. Figure 5 shows the variation of an adder area with respect to its width. These curves are plotted for Carry Save Adder(CSA), Ripple Carry Adder(RCA), and Carry Look Ahead Adder(CLA) architectures. Similarly, figure 6 shows the NAND gate equivalent area obtained for Shifter and Storage Registers. It can be interpreted that the area increases in a fairly linear manner with the number of bits for the data-path components, hence a cost function having linear dependence on the bits of functional and storage unit can be justified. Cost function is decided by the per bit area requirement of different data-path units. Relative weights per bit of the shifter and adder with respect to a register are approximately calculated based on the technology and architecture used. The cost function in add shift implementation is driven by number of adder, shifter and register bits.

$$Cost = W_a * N_a + W_s * N_s + W_r * N_r ;$$

Where :

$W_i$ : Weight per bit of unit $i$.

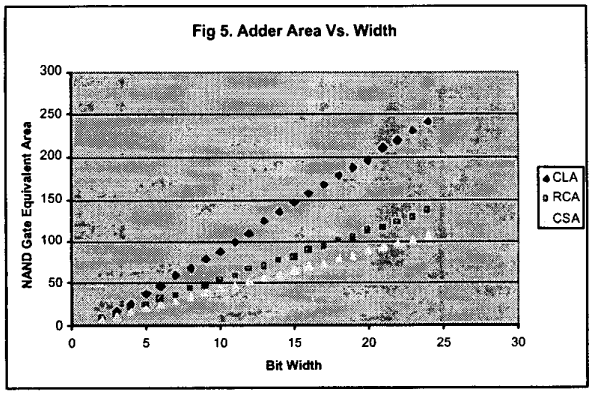
$N_i$ : Number of bits of unit $i$.

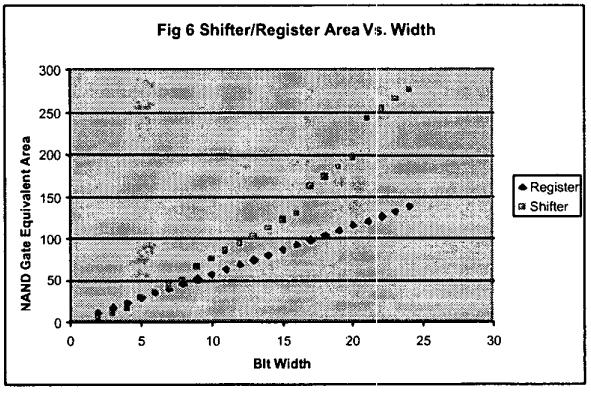
Here we haven't considered the interconnect in the optimization phase of the scheduling algorithm. We assume availability of Over the Cell (OTC) routing area, which is usually true in cell based ASIC designs. This is possible because of 5-6 or more layers of metallization in contemporary technology. We also exclude the mux area from the formulation of our cost function. Firstly, the contribution of the muxes in the area is quite small as compared to that of other components. Secondly, the mux area is directly proportional to its width and number of select signals. As our algorithm aims at minimizing the width of functional and storage units, it reduces the mux width implicitly and mux area is reduced consequently. We now present the scheduling algorithm.

## III(B). Precision sensitive scheduling algorithm:

***Problem*** : Schedule a given multi-precision DFG and allocate the functional and storage units with the aim of minimizing the total number of bits of functional and storage units with appropriate weight associated to them.

***Given*** :A DFG G=(V,E), where number of nodes 'V' signifies number of intermediate variables and directed edge $E_{i,j}=(v_i,v_j)$ depicts the dependency of node j on i.



***Objective*** : To schedule the DFG (i.e. to assign each computation to a control step) such that :

1. For each edge $E_{ij}=(v_i,v_j)$ : the node j is scheduled in control step later than the node i.

2. Number of scheduling steps are equal to the minimum possible for that DFG( the latency is not increased).

```
procedure multi_prec_HLS :

1.Given a multi-precision DFG, evaluate the bit precision
of each of the intermediate node.
2.Initial Schedule ="ASAP " or "ALAP" schedule.
3. Employ a cost based iterative improvement scheduling
technique with the cost function "COST".
End - procedure multi_prec_HLS

procedure COST (Schedule) :

Allocate storage units with "multi_prec_allocate".
Allocate the functional units with "multi_prec_allocate"
cost =  Ws*Ns+Wa*Na+Wr*Nr;
/*Where Wi(Ni) is normalized per bit weight(width)
associated with resource i*/
Return cost;
end - procedure COST
```

Figure 7: The HLS (Scheduling) Algorithm

3. The allocation results in optimized use of resources in terms of minimizing the total bits of functional and storage units. This further translates to minimizing the area requirement by the functional and storage units. The pseudo code for the algorithm is formally presented in figure 7. For scheduling we use KL [5] – based iterative improvement heuristic with either As Soon As Possible (ASAP) or As Late As Possible (ALAP) [7] scheduling being the initial solution. The objective of ASAP and ALAP is to obtain a slack within which a node can be moved, so that the latency bound is not exceeded. For each of the incremental scheduling step we perform complete allocation and binding of functional and storage units and the cost for the new schedule is computed with weights, as shown in the pseudo code.

## IV. Results:

In this section we report the results of implementation of our approach. The algorithms were implemented in C and were run on Solaris (SunOS-5) on Ultra-SPARC *IIi* CPU. The first column of table-1 represents the test cases. Second column lists the number of computation nodes(adds and shifts) in the DFG. In the first three sub-columns of the third column the adder(A1), shifter(S1),and register bits(S1) respectively are listed as obtained with the standard conventional scheduling, allocation and binding. The fourth sub-column of third column shows the resultant cost function computed with

| Test Cases | Comp. Nodes | Conventional Scheduling and Allocation | | | | Conventional scheduling and Precision. Sensitive Allocation | | | | | Integrated precision Sensitive Scheduling and Allocation | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Add bits $A_1$ | Sh. Bits $S_1$ | Reg bits $R_1$ | Cost $C_1$ | Add bits $A_2$ | Sh. Bits $S_2$ | Reg bits $R_2$ | Cost $C_2$ | Impr (%) | Add bits $A_3$ | Sh. Bits $S_3$ | Reg bits $R_3$ | Cost $C_3$ | Impr (%) |
| FIR-1 | 100 | 191 | 199 | 473 | 1107.4 | 176 | 191 | 412 | 1009.6 | 8.83 | 130 | 129 | 386 | 806.7 | 27.14 |
| FIR-2 | 85 | 154 | 121 | 380 | 824.0 | 147 | 121 | 334 | 767.2 | 6.89 | 113 | 89 | 334 | 660.2 | 19.88 |
| FIR-3 | 98 | 191 | 168 | 467 | 1048.4 | 190 | 173 | 435 | 1023.4 | 2.38 | 125 | 122 | 373 | 774.1 | 26.16 |
| FIR-4 | 118 | 154 | 128 | 478 | 934.0 | 144 | 127 | 446 | 884.9 | 5.25 | 118 | 92 | 400 | 739.0 | 20.87 |
| NTSC-8 | 40 | 65 | 62 | 167 | 373.1 | 63 | 62 | 153 | 356.0 | 4.57 | 49 | 45 | 149 | 301.4 | 19.23 |
| UVW-8 | 53 | 119 | 138 | 293 | 712.2 | 113 | 134 | 261 | 664.1 | 6.75 | 76 | 102 | 227 | 518.4 | 27.21 |
| UVW12 | 51 | 114 | 140 | 255 | 669.9 | 108 | 133 | 233 | 626.7 | 6.45 | 79 | 98 | 209 | 498.2 | 25.63 |
| DCT-8-12 | 152 | 245 | 198 | 577 | 1292.8 | 242 | 197 | 544 | 1253.5 | 3.04 | 180 | 133 | 508 | 1012.6 | 21.67 |
| DCT8-16 | 273 | 489 | 407 | 1252 | 2701.0 | 471 | 403 | 1135 | 2549.4 | 5.61 | 341 | 300 | 1040 | 2078.1 | 23.06 |
| IDCT6-12 | 102 | 183 | 162 | 443 | 1001.8 | 176 | 164 | 409 | 960.4 | 4.12 | 128 | 107 | 377 | 757.1 | 24.43 |
| IDCT10-12 | 286 | 411 | 349 | 1150 | 2379.7 | 399 | 341 | 1031 | 2228.5 | 6.35 | 314 | 226 | 966 | 1836.0 | 22.84 |
| IDCT 9-8 | 111 | 147 | 138 | 360 | 822.3 | 145 | 130 | 346 | 791.6 | 3.74 | 129 | 82 | 322 | 660.9 | 19.63 |
| | | | | | | | | | | | | | | Avg. Impr. | 23.14 |

**Table-1**

the weights associated with different resource-bits which signifies the area of implementation. The last sub-column lists the percentage improvement over the conventional approach The relative weights obtained with CLA as the adder architecture, as per the approach illustrated in section III are:

$W_a = 1.54$, $W_s = 1.71$ and $W_r = 1.00$.

The fourth column shows the respective results $(A_2,P_2,S_2,R_2)$ for precision sensitive allocation and binding with conventional scheduling. The last column shows the respective results $(A_3,P_3,S_3,R_3)$ for an integrated precision sensitive scheduling allocation and binding approach. The objective is to illustrate the overall gains obtained with precision sensitive approach in both scheduling and allocation phases of the high level synthesis. Note that in all the cases we have used the same KL based iterative improvement scheduling technique with the cost function being precision insensitive in the conventional approach and precision sensitive in the other case.

We have run our synthesis technique over a number of real life examples of FIR filters and DCT/IDCT computations[14] and color space conversions[13]. The inputs FIR1 to FIR4 in the first column represent FIR filters of 24 to 36 taps respectively. The inputs with the name IDCT(DCT) i-j represents 1-Dimensional IDCT(DCT) computation of i data points with the coefficient width j. NTSC and UVW are color space conversion matrix computations and the suffix indicates the coefficient width. We have achieved gains in area as high as 27.21% and average gains of 23.14% over the standard precision insensitive HLS procedures.

## V. Conclusion and Future Work

We have for the first time, to the best of our knowledge, addressed the problem of High Level Synthesis (HLS) of multi-precision DFGs. We have presented a precision sensitive scheduling algorithm. We have used an iterative improvement approach with cost function being formulated in terms of number of bits of arithmetic operators and storage units. An algorithm for register allocation and functional unit binding for variable precision arithmetic has also been proposed. We have also proposed an integrated HLS methodology to exploit the interdependence of scheduling, allocation and binding. Optimization ratios of as high as 27.21%(23.14% average) over the conventional fixed precision techniques establish the potential of our approach.

The size of a functional unit affects its area as well as performance. The system clock period is decided by the delay of the largest or most complex functional unit. We are planning to enhance this work by incorporating techniques such as scheduling high delay functional units over multiple cycles. This will lead to smaller clock periods and the system performance (throughput as well as latency) will improve.

## VI. References:


[1] M. Potkonjak, M.B. Shrivastav, P.A. Chandrakasan, "Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common sub-expression elimination", *IEEE Trans. Computer -Aided Design*, vol.15, Feb. 1996, pp.151-161.

[2] M.Mehendale, S.D.Sherlekar, G.Venkatesh, "Synthesis Of multiplier-less FIR filters with minimum number of addition," *Proceedings of the 1995 IEEE/ACM International Conference on Computer -Aided Design*, 1995, pp. 668-671.

[3] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, D. Durackova, "A New Algorithm For Elimination of Common Sub-expression", *IEEE Trans. on Computer -Aided Design of Integrated Circuit and Systems*, vol.18, No.1, Jan.1999.

[4] M. Ergcegovac, D. Kirovski, G. Mustafa, M. Potkonjak, "Behavioral Synthesis Optimization Using Multiple Precision Arithmetic", *ICASSP*-1998.

[5] K. H. Kernighan, S. lin, "An Efficient Heuristic Procedure for Partitioning Graph", *Bell Systems Technical Journal*, vol. 49, no. 2, Feb. 1970.

[6] Giovanni De Micheli, "*Synthesis and Optimization of digital circuits*", McGraw Hill Inc. 1994.

[7] Daniel D. Gajski, Nikil Dutt, Allen C. H. Wu, Steve Y. L. Lin , "*High-Level Synthesis : Introduction to Chip and System Design*" , Kluwer Academic 1992.

[8] A. Peleg, U. Weiser, "MMX technology extension to Intel architecture", *IEEE Micro*,vol. 16 No. 4, 1996.

[9] Tony M. Carter, "Cascade: Harware for High/Variable Precision Arithmetic", *9th Symp. on Computer Arithmetic*, 1989.

[10] Anand Pande, Sunil Kashide "*Hardware Software Co-design of DSP Algorithms*" M. E. thesis, Microelectronic Systems, Indian Institute of Science, Bangalore.

[11] P.G. Paulin, J.P. Knight, "Force Directed Scheduling for Behavioral Synthesis of ASIC's" *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 8, no. 9. June 1989.

[12] F.J. Kurudahi, A.C. Parker, "REAL:A Program for Register Allocationl", *Proceedings of the 24th Design Automation Conference*, pp. 511-516, 1990

[13] C. Gonzalez, R.E.Woods, "*Digital Image Processing*", Addison Wesley 1998.

[14] Alan V. Oppenheim, R.W. Schafer, "*Digital Signal Processing*", Prentice Hall of India, 1996.