Abstract
As we move towards several million transistors per chip it is desirable to move to higher levels of abstraction for the purposes of automated design of systems. Increasing performance of microprocessors in the marketplace is moving the balance between software and hardware. In this environment, it is necessary to adapt our tools to create systems, which encompass these fast microprocessors rather than compete with them. It is important to adapt other peripheral components such as sensors and RF circuits into our design methodology.

I. INTRODUCTION
How do we design with several hundred million transistors effectively and quickly? Hardware Software Co-design is said to provide the answer for designing such large systems. HW/SW Co-design [2,14,21,20,17,19,73,75] is a wide area of research consisting of simulation, validation, synthesis etc. In this seminar we will look at the synthesis aspect of co-design.

In the particular research area of synthesis, three distinct methods have emerged. The first method is the Co-processor method, which usually consists of a central off the shelf processor, and some application specific integrated circuits in order to speed up those parts of the program which are slow in the processor. The second method is the Application specific instruction processor method, in which a unique processor is designed and created for a particular application. This processor will have a unique instruction set and architecture, which is specifically tuned for the application. A third approach is to partition the task at hand into different components, and allocate each component to a separate processor.

Each of these three methods start with a variety of input specifications[15,4,5], such as software languages (C), Hardware Description languages (VHDL) [25] and specialized languages (Spec Charts). These languages are at times modified to include specific characteristics such as timing information and parallelism [29,30,6,41,42,63].

In discussing the three major methodologies the profiling techniques will be have to be critically considered. Further the quality of results from present estimation techniques will have to be examined. We will also need to look at the speedups achieved in Hardware Software Co-synthesis projects so far and compare the speedups with other traditional methods to see whether these Co-synthesis methods are really useful. We need to explore techniques to improve current speedups. We have to examine application programs and the data associated with them in order to improve the final result.

We also have to look at architectures that are presently being used and their weaknesses. Further, we need to explore architectures that could be used in the future with the current trends in microprocessor design etc. We have to analyse effects of emerging standardization such as Virtual Systems Interconnect (VSI).

II. LANGUAGES
Standard languages such as C and VHDL are very popular input languages. These have the disadvantage of either being software oriented or hardware oriented. Other specifications methods such as graphical or semi graphical methods are becoming popular. C++, FORTRAN etc are also used in specifying synthesis systems. These suffer because of the single thread nature and because no timing constructs exist. The advantages of these languages are that these are widely used and thus have good support and several analysis tools. Several extensions are available for these languages such as PVM, P4, HPF, and MPI.

There are several in house languages such as Jade and SAM which are probably more suitable but have little or no support. Languages which are state charts [22] based such as SPECHARTS are useful in describing systems are becoming popular, but are not widely available yet.

III. PARTITIONING
As we rapidly move towards a time of low cost, high-speed microprocessors and cores [1,13,26,50,51,52,53,67], the line between software and hardware is changes from month to month. What someone would have said with certainty should have been implemented in hardware just a year ago, is probably implementable in software for a fraction of the cost without any sacrifice in performance today. In light of this
transience in the state of the art we have to critically look at what are the alternative we can research with certainty over the next few years. A research project started today will often take a few years to complete, and can be totally irrelevant to the market place, leaving the researcher feeling ineffective.

There are several types of architectural models, which use both processors and ASICs. Models included a single processor and a single ASIC; single processor with several ASICs, several processors and several processors with several ASICs[54]. All systems, which automatically synthesize circuits based on these models, include an estimation system and a partitioning system. The estimation system allows the quick evaluation of alternative partitioning solutions in the design space. Partitioning solution allows the total task to be optimally shared by processors and ASICs, according to a given set of criteria, be it speed, cost or low power.

Partitioning algorithms described are usually very effective and fast. However these tools depend on estimation tools and profiling tools for their final partition which then makes it quite unreliable.

IV. PROFILING TOOLS

Profiling tools [74] are a necessity to get information on how long a particular segment of code takes to execute and how many times a loop (with indeterminate loop counts) is executed. Tools such as GPROF are not accurate and give widely varying results. Execution graphs have been used which have also been shown to be not accurate.

The major reason behind these inaccuracies is the fact that the architecture greatly influences the final result. Since the architecture is not known at the beginning, the profiling will inevitably be wrong.

Simulation [18, 47] and emulation tools can be used to accurately predict performance. However, these tools are slow, and are only useful at the latter stages of the design cycle.

V. ESTIMATION TOOLS

Estimation tools have been notoriously ineffective in the past. Three of the most used estimation tools have been: profiling tools to estimate the time taken in software for a given length of code (see last section); area estimation tools to assess the probable size of the ASIC when finally implemented; and the execution time estimation tools for ASICs which, estimate the execution time of an ASIC.

The tools used to estimate the size of area can be extremely error prone, particularly since it is difficult to estimate the interconnection area. The error in estimate can be very costly since the cost of chips is stepped rather than a linear function. For example a chip 2.01x2sq.mm chip can cost a great deal more than a 2x2sq.mm chip. This situation can possibly improve as more and more pre-fabricated cores are used in design, which would then reduce the total amount of unknown interconnection area.

The time taken for a particular ASIC to execute is another estimate, which is difficult without the final layout, since the clockwidth cannot be predicted. Often predictions are based on the number of cycles, but this is almost useless without the clockwidth information.

The other important consideration is the power estimation. This estimate at the present time is very sketchy, though there are some experts who given a particular process, number of gates, clockspeed etc can estimate the power to within 10% of its real value.

VI. FUTURE OF COSYNTHESIS

There are several areas where real progress has to be made. These are accurate but quick profiling and estimation techniques, better specification languages, improved communication methodologies, automated architecture synthesis and testing of such systems [3,8,31,34-37,28,54-56,62]. Heterogeneous systems, which are automatically synthesized, will be a major research area in the near future [10,12,57,59,66,70-72]. Compiler optimization for simple systems is now available [7,11,16,23,24,32,33,38-40,43-46,48,49,52,58,60,61,64], but will have to be modified and improved for heterogeneous systems.

There are other areas where little or no work has been carried out. For instance there is little work in automatically integrating sensors to digital systems, or interfacing RF circuits to digital systems, or integrating analog drivers to embedded systems.

References


