CHAPTER 1

INTRODUCTION

This chapter emphasize the significance and characterization of VLSI Systems, high level synthesis and scheduling algorithms. A brief review of literature pertaining to the present work is also presented.

1.1 VLSI TECHNOLOGY

As the growth of VLSI technology, the digital circuit design becomes more and more complex. In the early 2000’s, VLSI technology reached densities of over tens of millions transistors of random logic per chip. Figure 1.1 shows the expected number of transistors on various types of chips assuming a constant die size of 400 mm$^2$. As one example it predicates by 2011, 3.2 billion transistors ASIC will be feasible in VLSI technology.

![Figure 1.1 Number of transistors on various types of chip (400 mm$^2$) as projected by the Semiconductor Industry Association (SIA)](image)

(Source: The Electronics handbook, Jerry C. Whitaker)
The device trends in volume and size, increase in device count per chip and shrinkage of feature size are shown in Figure 1.2 more than one-million fold increases in the device count has been achieved these 40 years leading to almost the same increase in processor performance. This has been driving enormous development of electronics and information technology.

(Source: Trends in Integrated Circuits Technology, Saraswathi)

**Figure 1.2 Trends in device count/chip and features of MOS device**

For such complexity, it becomes a difficult task to write RTL description for the system and would be tougher for system verification. There are also many possible structures for the system. Which are the optimal ones? How to limit the design space rapidly? How to find or approximate Pareto points in the Design Space quickly? Moreover, VLSI technology has reached a maturity level. It is far beyond the design and verification capacity (Figure 1.3).
Today time-to-market is given the same or more emphasis than the area and the speed. The design purpose is to have a design satisfying the specification in the shortest time. A delayed design would lose the market and profit. The industry suffers much pressure on the time-to-market and they are willing to accept a sub-optimal design developed in the shortest time, which is more cost efficient.

(Source: www.sophen.com)

Figure 1.3 The forecast of the productivity

Figure 1.4 Revenue vs. Time-to-Market
Figure 1.4 shows the relation between the revenue and the time-to-market. The delayed entry would result in loss or profit. If the design is delayed by market window time, the customer would just skip to the next generation design. There would be no profit at all. And also as the complexity of the system increases, how to evaluate and simulate the system efficiently becomes an urgent problem. About 75% work load in a system design is for verification.

To try to solve all these problems, there is a trend toward a new design methodology – to design automatically on more abstract levels. As for more abstract level, the description and the tradeoff for the system are much easier. To the request of this new design methodology, High-Level Synthesis tools are developed to satisfy the demand. High-level synthesis is a translation process from behavioral description into RTL description in an automatic or semi-automatic way. There are several languages used for behavioral description. Behavioral VHDL/Verilog, Matlab, ANSI-C and SystemC and also corresponding tools to translate these descriptions into RTL descriptions. Behavioral Compiler (Behavioral Compiler VHDL User and Modeling Guide - Version -U2003 2003) is the first successful commercial High-Level Synthesis tool. It takes behavioral VHDL/ Verilog description as input and output the synthesizable RTL description, such as SPARK (SPARK: a High-Level Synthesis Framework) for ANSI-C, Cocentric (Cocentric), Cynthesizer (Cynthesizer User’s Guide) for SystemC, Icarus Verilog Open Source Compiler and Synopsys.

1.2 HIGH LEVEL SYNTHESIS

High-level synthesis (Chang & Pedram 1999) allows the designer to write high-level (above RTL) behavioural descriptions of a design that can be automatically converted into synthesizable RTL code (Raghunathan et al. 1998). High level code describes the operation on different data types of a
hardware design without specifying the required resources and schedule of cycles. Thus, high-level synthesis offers the advantage of automatic handling of the scheduling and synchronization issues of a design, which helps in faster architectural exploration leading to reduced design time and increased productivity. Figure 1.6 and Figure 1.5 shows the typical high-level to gate-level design flow used in the hardware industry to generate hardware designs. Gate-level designs are further synthesized using other downstream tools to generate ASICs or FPGAs.

(Source: Gajski-kuhn, An Overview of today’s high-level synthesis tools)

Figure 1.5  Gajski-kuhn Y-chart for HLS design flow

Today’s VLSI Technology allows companies to build large, complex systems containing millions of transistors on a single chip. To exploit this technology, designers need sophisticated Computer Aided Design (CAD) tools that enable them to manage millions of transistors efficiently. The design complexity in handling the millions of transistors is reduced to a greater extent by High Level Synthesis (HLS). The High Level Synthesis is
the powerful and effective tool to achieve the hardware design of modern day digital applications.

Figure 1.6 High-level to gate-level design flow

The progression of Very Large Scale Integration (VLSI) system, improving in various fields such as Communication, Smart phone, Biological application, Consumer electronics and Weather forecasting. These growing trends in application, needs the design of hardware system intend only to particular applications. This will lead to an efficient execution of the given task by utilizing the specific hardware design. According to Moore’s law, the performances of the integrated circuit will double every eighteen months. On the other hand, System on Chip (SoC) (Benini et al. 2005), has emerged as a result of continuing improvement in VLSI system. The application idea is still more extended by incorporating the advantages of System on Chip (SoC). The
performance of the SoC system will degrade as there is a limitation in bus architecture such as wire-delay, power consumption and large area due to bus contention. In the modern era of System on Chip (SoC), the application’s power, performance, delay and area are dominated by the process of on-chip communications (International Technology Roadmap for Semiconductors 2009). Therefore, the architecture for the on-chip communication in SoC is carried out by different architectures such as Network o Chip (NoC) (Benini et al. 2005), bus matrices (Renshen et al. 2011) and overview of various architectures is shown in (Milica & Mile 2006). The following Figure 1.7 shows the High Level Synthesis flow from behavioral specification to Register Transfer Level.

![Figure 1.7 Steps of HLS](Source: Electronic Design Automation for IC System Design, Luciano)
Starting from the high-level description of an application, an RTL component library, and specific design constraints, an HLS tool executes the following tasks: compiles the specification, allocates hardware resources (functional units, storage components, buses and so on), schedules the operations to clock cycles, binds the operations to functional units, binds variables to storage elements, binds transfer to buses and generates the RTL architecture. Tasks allocates hardware resources through binds transfer to buses are interdependent, and for a designer to achieve the optimal solution, they would ideally be optimized in conjunction. To handle real-world designs, however, the tasks are commonly executed in sequence to manage the computational complexity of synthesis.

1.2.1 Need for HLS

In 1990s, the HLS tool was declared as failure model for various issues; the HLS with high quality and performance are in need for the following reason.

**SoC with Embedded Processor**: The modern embedded system involves number of software elements along with custom logic and memories on a single chip, micro processor and digital signal processor. The C/C++ programming of the customized hardware logic and embedded software possessed by automated flow of the HLS tool made them a perfect match for System on Chip (SoC).

**Enhancing the Design Yield by reusing the Behavioral IP**: The reuse of behavioral IP in behavioral synthesis is the added advantage apart from the reduction in line count. This IP can be realized in different technologies are re-targeted to diverse realization, whereas the IP in RTL have the constraints of fixed micro-architecture.
**Heterogeneous SoCs and Accelerator Development:** The processor with multi programmable ability has the constraints of power which is overcome by accelerators based on custom architectures. And nowadays this accelerator was incorporated in much System on Chip (SoC) and Chip Multi-Processor (CMPs). By 2024, the accelerators present on-chip will reach 3000 as predicted by ITRs (International Technology Roadmap for Semiconductors 2009). The performance of the design is increased by using custom architecture in FPGA whereas the degradation in performance was the result of using soft processor. Again, this custom logic is more suitable to High Level Synthesis.

**High Level of Abstraction Rises Due to Massive Silicon Capacity:** The improvement in design productivity and control over complexity is achieved by using design abstraction. As an example from NEC (Wakabayashi 2004), RTL code with 300K line is needed for 1M-gate design, which is hard hitting for the manual designer. The reduction rate of 7X-10X is achieved by utilizing the SystemC, ‘C’ or ‘C++’ as a programming language for specification of high level design. The code complexity in the design is reduced by reducing the line of code to 30K - 40K with the help of high level synthesis (Wakabayashi 2004).

**Demand of System Level Verification:** The system level verification (Ghenassia 2005) is approached by the method of Transaction-level modelling with a programming language called as SystemC (IEEE 1666 TM-2005 Standard for SystemC 2005). The TLM based on SystemC is commonly used by designers which facilitate the designer with verification based on functionality, development of embedded software and finally the modelling of architecture. This would lead to the HLS solution based on SystemC. In today’s industry the recording of RTL manually is replaced by RTL code generated automatically by HLS tools.
Reactive Time-to-Market: the inadequacy in full custom ASIC (i.e. long time for designing chips and manufacturing cycles) is overcome by the FPGA. Faster time-to-market is achieved by the FPGA. The tradeoffs in power, cost and performance is overrated by reducing the time to design. This choice of tradeoffs between performance and design time is handled over to the designer in modern tools.

Inefficiency of Verilog and VHDL: Application in High Performance Computing (HPC) such as bioinformatics, video and image processing, scientific computing application and financial analytics are made familiar with advancement in the reconfigurable computing of the FPGA. The application developer finds difficult in coding the HPC in VHDL or Verilog. On the other hand, the HLS tools provide the feasibility of programming the application using ‘C’, Matlab or ‘C++’.

Inexpensive Formal Verification: The success in the first tape-out is the vital problem for the ASIC designer. From (International Technology Roadmap for Semiconductors 2009), over $1M is utilized to manufacture the IC in nanometre technologies. Till now mature HLS tools for formal verification of ASIC are not evolved properly. Moreover the verification is limited to SoC having multi-million gates. On the other hand the simulation coverage for FPGA is possible with higher degree. The manufacturing cost can be reduced to larger extent for more number of design iterations.

Ultimate for Synthesis based on Platform: The HLS tools can support a Quality of Result (QoR) and design methodology based on platform (Keutzer et al. 2000). The above said feature is possible because, the predefined IP like embedded processors, embedded memories, embedded system bus and arithmetic units are entrenched in the modern FPGAs. In FPGA platform the modelling of the predefined logic is planned ahead, as a result of it the design can be done efficiently. The research interest in the field of HLS tools made
themes available for various application domains like image processing (Denolf et al. 2009), data analysis in cosmology (Kindratenko & Brunner 2009), application in aerospace (Pingree et al. 2008) and wireless system (3G/4G) (Guo et al. 2006). Also, Xilinx Inc. featured their user by embedding the solution of HLS in their DSP Development Kit (Avnet Spartan-6 FPGA DSP Kit) and Video Development Kit (Xilinx Spartan-6 FPGA Consumer Video Kit).

**Early Evolution HLS:** In earlier stages, the HLS tools are developed to target the ASIC rather than an FPGA. The seed was laid by the researchers at the University of Carnegie Mellon in 1970; they develop an HLS tool called CMU-DA (Director et al. 1982, Parker et al. 2004). This tool uses the Instruction Set Processor Specification (ISPS) language (Barbacci et al. 2005) to specify the design in behavioral level. The behavioral design is converted to intermediate data called as value trace (Snow et al. 2000) afterwards converting it to RTL design. Many techniques in code-transformation such as elimination redundant sub-expression, extraction of common sub-expression, code motion, propagation of constant and elimination of dead-code made added advantages to the compiler. On the other hand, the features like controller generation, allocation of the datapath, selection module and hierarchical design support were offered at the synthesis engine.

**1.2.2 HLS Evolution after 2000**

The next generations of high-level synthesis tools were developed in 2000 by both industry and academia. The design was made possible with many tools by using the programming languages like C/C++. The application developer and system designer are more convenient with tools, operating with ‘C’ like language rather than HDL languages previously used. The vital character of synthesis tools such as optimization and parallelization are made easy for the designer, using the software compiler made by ‘C’– like
programming languages. However, the programming language like ‘C’ has the shortcomings that it will suitable only for the microprocessor that runs on sequential software. Indeed (Sanguinetti 2006, Edwards 2006) explain the ongoing contest of choosing the ‘C’ or HDL language for HLS. The contesters mainly focus on the HDL language, since the ‘C’ language has the following constraints such as specification of concurrency, timing, synchronization and accuracy of bits, which is decisive to hardware design. Other than this, the language construction is complex since it involves polymorphism, pointers, recursion and managing the memory dynamically, which would lead to the synthesis complication. The constraints of ‘C’ language are overcome in modern HLS tool with extended ‘C’ language, such as SpecC (Gajski et al. 2000, HardwareC (Ku & De Micheli 1999) and Handel-C (2007). These languages make the tools, feasible to the user to specify the compiler directives and libraries free from timing, concurrency and other constraints. The two major advantages of these types of approach is that, the C/C++ compiler is enough to compile the input without change in the compiler and the co-verification/co-design of the software and hardware are not in need of re-writing the code. Dissimilar to the design based on ‘C’ as input, the other tools which accept the inputs other than the ‘C’ languages are as follows Matlab (Haldar et al. 2009), BlueSpec (BlueSpec Inc.) and Esterel (Edwards 2005). And in nowadays, the HLS tools are developed by keeping the target platform of FPGA in mind.

The reconfiguration and re-programming feature of the FPGA made them a striking candidate for many application likes video and image processing, signal processing, communication and in HPC. So the modern HLS tools are developed to target the FPGA, such as, Impulse C (Pellerin & Thibault 2005), Streams-C compiler (Gokhale et al. 2008), SPARK (Gupta et al. 2004), Trident (Tripp et al. 2007), Nallatech developed DIME-C (DIME-C user guide), Altera developed C2H (Jios II C2H compiler user
guide, version 9.1 2009), CASH (Budiu et al. 2002), ASC (Mencer), GAUT (Coussy et al. 2008), Mentor Graphics developed Handel-C (Handel-C language reference manual 2007) and ROCCC (Villarreal et al. 2010). For example, the floating point in the FPGA can be effectively realized using Trident compiler.

The applications based on Digital Signal Processing (DSP) are implemented effectively using tools such as ASC, Streams-C, ROCCC, Impulse C and GAUT. Finally, the application that has resource constraint (i.e. need of optimized hardware) can utilize the work of Technology driven High Level Synthesis (THLS) (Joseph et al. 2007). As of 2012, the HLS tools based on ‘C’ language and also the other language are as follows NEC’s Cyper Workbench (Wakabayashi et al. 2008), Catapult C (Bollaert 2008) developed by Mentor Graphics, Icarus Verilog (iverilog), C-to-Silicon compiler (Bailey et al. 2010) developed by Cadence, Synphony C (Synopsys) developed by Synopsys, Autopilot (Zhang et al. 2008) from AutoESL’s.

1.3 SCHEDULING ALGORITHMS

Scheduling is defined as that step in high-level synthesis in which the operations are grouped into control-steps based on their types and dependencies in such a way that the operators in the same control step could be executed simultaneously. A wide variety of approaches exist in efficient scheduling which are directed at either reducing the total time of execution or minimizing the number of resources needed for the design. Broadly, these approaches could be classified into four categories: Basic scheduling, time constrained scheduling, resource constrained scheduling and miscellaneous scheduling.

The control and data flow graphs depict the inherent parallelism in a design, based on which, each node could be assigned a range of control
steps. Most of the scheduling algorithms require the earliest and the latest bounds that define the range of control steps for each node in the CDFG. Two simple schemes that are widely used to determine these bounds are called the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) algorithms.

The ASAP algorithm begins with scheduling the initial node, i.e. nodes without any predecessors, in the first time step, and assigns the time steps in increasing order as it proceeds downwards. The algorithm is guided by the simple principle that a particular node can be executed only if all of its predecessors have been executed. Ignoring resource constraints, this algorithm gives the least number of control steps required for the design, and hence, could be used for near-optimal micro code compilation (Paulin & Knight 1989).

The ALAP algorithm is analogous to the ASAP scheme, except that the operations here are intentionally postponed to the latest possible control step. The algorithm begins at the bottom of the CDFG, i.e., with nodes that have no successors, and proceeds upwards to nodes that have no predecessors. This algorithm gives the slowest possible schedule for a given design.

1.3.1 Time-Constrained Scheduling

The time-constrained scheduling approach is often adopted for designs targeted towards applications in real-time systems, like the digital signal processing systems, which are often limited by the response time. Here the main objective would be to realize the design with minimum possible hardware while meeting the time constraint. Time constrained scheduling is usually implemented using three different techniques: Mathematical programming – Constructive heuristics – Iterative Refinement.
Integer Linear Programming the ILP method is a mathematical formulation of the scheduling problem, which applies a branch-and-bound search algorithm with backtracking to find the optimal schedule.

\[ P(x, z) = P(x) \text{ whenever } P(y, z) > 0 \]

The ILP approach begins with finding the earliest \( (E_k) \) and the latest \( (L_k) \) time-bounds for each operation using the ASAP and ALAP algorithms respectively.

Such a formulation could be extended to further include resource and data dependency constraints using the equations,

\[(\text{Source: FDS for Behavioral synthesis})\]

**Figure 1.8 Force directed scheduling approach**

where \( p \) and \( q \) are the control steps assigned to the operations \( x_i \) and \( x_j \) respectively.
One major drawback of the ILP formulation is that its complexity increases rapidly with the number of control steps. For a single additional control step, \( n \) additional \( x \) variables have to be considered. The ILP approach is computationally exhaustive and hence, can be applied only to very small problems.

One other approach for time constrained scheduling is a heuristic method, called the Force directed scheduling. This algorithm tries to reduce the total number of functional units used by uniformly distributing the operations of the same type over the available control steps.

1.3.1.1 Force directed scheduling algorithm

The force-directed scheduling (FDS) heuristic is a well known heuristic for scheduling with a given timing constraint. Here present a simplified version of the FDS algorithm. The main goal of the algorithm is to reduce the total number of functional units used in the implementation of the design. This objective is achieved by uniformly distributing operations of the same type into all the available states. This uniform distribution ensures that functional units allocated to perform operations in one control step are used efficiently in all other control steps, which leads to a high unit utilization rate.

The FDS algorithm relies on both the ASAP and the ALAP scheduling algorithms to determine the range of control steps for every operation (\( m_{\text{range}}(o_i) \)). It also assumes that each operation \( o_i \) has a uniform probability of being scheduled into any of the control steps in the range, and probability zero of being scheduled in any other control steps. Thus, for a given state \( s_j \), such that \( E_i \leq j \leq L_i \), the probability that operation \( o_i \) will be scheduled in that state is given by \( p_j(o_i) = 1/(L_i - E_i + 1) \).
These probability calculations can be illustrated using the example from the above Figure 1.8, with the ASAP (E_i) and ALAP (L_i) values from Figure 1.9 used in the calculations. The operation probabilities for the example are shown in Figure 1.9. Operations o_1, o_2, o_5, o_7, and o_8 have probability values of 1 for being scheduled in steps s_1, s_1, s_2, s_3, and s_4 respectively, because the s_i value is equal to the s_i value for these operations. The width of a rectangle in this Figure 1.9 represents the probability \((1/(L_i - E_i + 1))\) of an operation getting started in that particular control step. For example, operation o_3 has a probability of 0.5 of being assigned to either s_1 or s_2. Therefore, the value of \(p_1(o_3) = p_2(o_3) = 0.5\).

(Source: FDS for Behavioral Synthesis, Paulin)

Figure 1.9 Force directed scheduling algorithm

In each iteration of the FDS algorithm, one operation is assigned to its control step based on the minimum expected operator costs. If there are
two possible control step assignments with close or identical operator costs, then the above algorithm cannot estimate the best choice accurately.

The method is called "constructive" because a solution is constructed without performing any backtracking. The decision to schedule an operation into a control step is made on the basis of a partially scheduled DFG; it does not take into account future assignments of operators to the same control step. Most likely, the resulting solution will not be optimal, due to the lack of a look-ahead scheme and the lack of compromises between early and late decisions. The solution can be optimized by rescheduling some of the operations in the given schedule.

1.3.2 Resource-Constrained Scheduling

Resource constrained scheduling algorithms are used in applications where the design is restricted by the silicon area. The goal of these algorithms is to minimize the number of control steps while satisfying the resource constraints. The schedule is built one operation at a time, so that the resource constraints and data dependencies are not violated. The total numbers of control steps are minimized in such a way that the number of operations scheduled in any control step does not exceed the number of FUs available.

Two popular approaches for scheduling operations with resource constraints include list-based scheduling and static-list scheduling.

List-based scheduling is based on including resource constraints in the ASAP algorithm. A priority list of ready nodes is maintained, and each such list is associated with a priority function that resolves any resource conflicts. A ready node is a node whose predecessors have already been scheduled. The algorithm proceeds by first scheduling operations with highest
priority while the lower priority operations are deferred to later control steps. At every step, the successors of a scheduled operation are added to the priority list of ready nodes.

The efficiency of such a list scheduling algorithm depends mostly on the priority function employed. A simple priority function could be chosen as to assign a priority that is inversely proportional to the mobility of the operation, and thereby, ensure that operations with large mobility are deferred to later control steps since they could go into more number of control steps. One major drawback of the list-based scheduling is the increased time and space complexity because of the several lists that have to be maintained dynamically.

1.3.2.1 Static list scheduling algorithm

The static-list scheduling is based on building a single list of operations statically, as opposed to the normal list-based scheduling, where the list grows dynamically. The ASAP and ALAP algorithms are applied initially to find the mobility range for each operation. The operations are sorted in ascending order based on their greatest control step assignment, and then, the operations with the same greatest control step value are sorted in descending order of their least control step value. The operations are then scheduled sequentially in the descending order of their priority. The operations that cannot be scheduled in a control step due to unavailability of resources are deferred to later control steps.
1.3.3 Other Scheduling Approaches

Apart from the previously discussed scheduling algorithms, several other approaches, like the Simulated Annealing, have been successfully used to solve the scheduling problem.

In the Simulated Annealing based approach (Devadas & Newton 1989), scheduling is treated as a placement problem, where the operations are to be placed in a two-dimensional table of control steps versus available functional units. The algorithm begins with an initial placement of operations, and iteratively modifies the table by displacing an operation. The new schedule is evaluated based on the cost of displacement, and is accepted with a probability, even when it may not be better from the previous one, in order to overcome local minima in the solution space. The simulated annealing approach is, thus, suitable for obtaining globally optimum solutions, but, requires long execution times for finding them.
Another approach is the Path-based scheduling (Camposano 1991), which is based on minimizing the number of control steps needed to execute the critical path in a CDFG. Initially, all possible paths of the CDFG are extracted and scheduling independently and later these schedules are combined to get the final schedule. The algorithm transforms the problem of introducing minimum control step constraint into a clique-partitioning problem. A clique partitioning solution would indicate like minimum overlapping of intervals in a given path.

Static List Scheduling algorithm first creates a single static large list before scheduling preventing dynamic growth. It uses ASAP and ALAP to obtain least (LCS) and the greatest possible control step assignments (GCS) for each operation. All operations are sorted in ascending order with GCS and descending order with LCS to form priority list. Operations are scheduled sequentially with highest priority, when the limit for the number of resources is reached rest of the operations is deferred to later control steps. Scheduling mainly depends upon the type of operation. For Ex: A low priority addition operation can be even scheduled in spite of high priority multiply operation if adder unit is available and if its predecessors are already scheduled.

List based scheduling is a generalization of the ASAP algorithm with the inclusion of resource constraints. These maintain a priority list of nodes called ready nodes whose predecessors have already been scheduled. Scheduling a node in a control step makes its successor nodes as ready nodes which could be added in priority list.

1.3.4 Scheduling the CDFG

Scheduling and allocation are two important steps in the synthesis process after translating the algorithmic description into an internal
representation. Scheduling is the process of assigning a control step to each operation of the Control and Data Flow Graph. A control step refers to the clock cycle in which the corresponding operation would be executed. Allocation is the process of assigning various functional units to operations, storage units to values, and buses to data transfers. A control unit could then be synthesized to synchronize the execution of operations based on the way that operations are scheduled and the hardware units allocated.

A number of scheduling and allocation methods have been proposed so far. It enumerate certain quality measures that could be used to evaluate the performance of scheduling and/or allocation algorithms: The quality of the solution produced; an optimal or a sub optimal design, The complexity of the algorithm; the CPU runtime, The solution space exploration, The possibility of handling large applications efficiently, The controllability of the synthesis process through designer constraints: area, delay, design rules, power consumption, etc, The possibility of predicting, with a maximum degree of accuracy, the previous parameters at a high level.

A complete formulation of the scheduling problem is given as a mathematical description which takes into account almost all the area parameters, without any increase in the complexity. The cost functions used in most of the other scheduling approaches impose a restriction on the solution space by fixing the Functional Unit (FU)s performing each type of operation. With the scheduling approach adopted in our synthesis system, module selection could be performed after scheduling in order to better optimize the design that is to be generated.
1.4 SCOPE AND OBJECTIVES

The automation of design process has been deemed necessary by the increasing complexity of the designs and the decreasing marketing-time requirements of the design market. Shifting the design process to higher levels of abstraction has been the motivating factor for several research works in the High-level synthesis phase. Despite the availability of several tools for synthesizing behavioral descriptions of designs, their application in research work is quite limited since most of them are commercially-oriented tools.

Despite the limited success of early generations of commercial high-level synthesis tools, we believe that the recent wave of research and development will likely lead to a much wider acceptance of high-level synthesis in the near future. The market needs for high-level synthesis have been growing in the nanoscale IC technologies. The reason for this is the following.

Larger silicon capacity and cheaper transistors: In the 1990s, when the first generation of commercial high-level synthesis tools was introduced, the design complexity was still manageable at the RT level, especially with other productivity boosters such as IP reuse. With continued scaling, these techniques become insufficient, and a paradigm shift toward a higher level of abstraction is increasingly desirable. At the same time, as transistors become cheaper, designers are more likely to accept a slight overhead in chip area when design productivity is enhanced by switching from manual RTL coding to high-level synthesis.

This dissertation is to explore the High-Level synthesis design methodology in various aspects. Here the HLS process use a FIR filter and
others as a design example to go through the whole design flow – from behavioral description, RTL synthesis to FPGA implementation. The qualities of results are compared with the result using other RTL methodology. The capability of several different High-Level Synthesis tools is also investigated and the results of several HLS tools are compared.

1.5 ORGANIZATION OF THESIS

The rest of the thesis is organized as follows: We enumerate some of previous works related to this field and detailed descriptions about Hardware Description Language, different optimization techniques in HLS, FPGA and different availability of High Level Synthesis Tools related to the research explained in chapter 2. Motivation about the research work about the identification and detailed research literature survey described in chapter 3. Chapter 4 gives a brief overview of the parallel computing and descriptions about the AMBA AHB bus matrix architecture and proposed logic using Icarus Verilog and results are discussed. Chapter 5 describes the proposed scheduling algorithm and results of the work compared and discussed. Experimental results obtained upon some of the standard high-level synthesis benchmark circuits are presented in chapter 5. Finally, the concluding remarks and future scope in chapter 7.