Generation and Optimization of Data Path in High-Level Synthesis

GE, Liangwei

情報生産システム工学専攻 最適化技術研究

2008年11月
The Moore’s Law describes that the number of transistors on a single chip doubles approximately every two years. This law has been valid for about five decades and is expected to continue for another ten years at least. As the integration density grows exponentially, ability to manufacture complex chips outpaces the ability to design and verify them. Consequently, there is a growing demand of moving to higher levels of abstraction and an automated approach to synthesize circuits from these abstractions.

High-level synthesis (HLS) is the core technology in raising abstraction levels, which is the automated conversion of circuit behavior (described in C) to register-transfer level implementation (Verilog or VHDL code). By moving to higher level, designers handle fewer, more abstract components and the HLS EDA tools fill the gap between manufacture and design abilities. However, HLS has not been widely accepted in the engineering practice so far, which is primarily due to the quality loss when compared with manual designs and the various limits on using HLS. To overcome these problems, some key issues need to be further improved. These issues include compiling, more accurate estimation, layout-aware scheduling/binding, HLS for reliability, support of floating-point arithmetic, etc. This thesis addresses two essential issues in HLS – scheduling and support of floating-point arithmetic.

Scheduling is the assignment of operations in the circuit behavioral specifications to clock cycles while satisfying a set of constraints (e.g. resource, delay). By determining the activities in each clock cycle (e.g. operation executions and data transfers), scheduling adds more details to the circuit temporal behavior, making the behavior cycle-accurate. Therefore, scheduling affects the final circuit on many aspects. As VLSI design enters the nano-scale era, new challenges arise (especially from the physical level), like noise, power, interconnection, etc. These challenges cannot be solved by layout alone and require combined efforts from all levels. Thus, new scheduling algorithms that are capable of addressing these problems need to be developed. However, scheduling under the conventional design constraints (i.e. resource and dependency constraints) is already an intractable (NP-hard) problem. Adding new physical-level objectives makes the problem even harder. Therefore, novel, aggressive techniques are needed to cope with these challenges.

Support of floating-point arithmetic is another essential issue in HLS. One of the challenges in converting the C-like behavioral description to RTL implementation is the incompatibility of number systems. The behavioral level supports a wide range of floating-point functions, like $1/x$, $\sqrt{x}$, $\log(x)$, $\sin(x)$, and $e^x$. The register-transfer level, on the other hand, is closer to hardware and assumes bit-vectors as fixed-point numbers by default. Even the single/double precision floating-point data type is not
supported in the RTL languages (i.e. Verilog and VHDL). Currently, almost all HLS EDA tools cannot synthesize floating-point arithmetic. Widely used functions (e.g. \(1/x\), \(\sqrt{x}\), \(\log(x)\), \(\sin(x)\), etc.) need to be manually implemented by fixed-point computations, which is slow and labor-intensive. Therefore, an automated translation of general floating-point functions to RTL implementations will have very practical value to HLS.

This thesis can be roughly divided into two relatively independent parts. The first part focuses on scheduling algorithms. The latter deals with floating-point function synthesis. More detailed summaries of each chapter are listed as below.

Chapter 1 [Introduction] firstly gives an overview of the high-level synthesis problem (the definition, synthesis flow, and previous works of HLS are shown). Then, Chapter 1 describes some key issues in HLS that need further improvement. Finally, Chapter 1 shows the contributions of this thesis – the proposal of the max-flow scheduling algorithm and a floating-point unit synthesis method. The importance of the two issues and the reasons why they are selected are explained as well.

Chapter 2 [Max-Flow Based Scheduling Algorithm] focuses on an essential issue in high-level synthesis – scheduling. To handle the emerging challenges of nano-scale IC design, scheduling algorithms that are able to optimize complex objectives (such as physical-level problems) early in the design process are needed. However, the widely used scheduling algorithms are constructive methods and cannot effectively handle the interaction between operations, which is required to optimize many physical objectives. Chapter 2 proposes a new scheduling algorithm: *max-flow scheduling* that formulates operation assignment as a matching problem on bipartite graphs. By defining the edge weight of the bipartite graphs, the proposed algorithm is capable of assessing all operation assignment simultaneously and refining operation interactions from a wider, systematic perspective. The ability to consider the cost of each operation assignment gives the max-flow scheduling algorithm more flexibility to cope with the various constraints and objectives in practical HLS systems. To demonstrate the ability of the new algorithm, Chapter 2 chooses the IC peak current reduction problem, which is traditionally solved during layout (e.g. by inserting decoupling capacitors, sizing up wires, packaging, etc.). Experiment shows that with the identical resource usage and schedule latency, the proposed algorithm more evenly distributes multiplications (assumed to consume much more current than other operations) along the clock cycles. The resultant schedule draws current more steadily from the supply (variance reduced by up to 65% under rough estimation), which mitigates the voltage fluctuation in the on-chip power distribution network. The results verify the flexibility of the proposed algorithm on optimizing complex objectives. This advantage is vital to nano-scale IC
design because optimizations made at higher level tend to have greater impact on the circuit than those made at lower level.

Chapter 3 [Exploration of Schedule Space by Random Walk] improves the works of Chapter 2 by proposing the enhanced max-flow scheduling. As is known, scheduling in general is an NP-hard problem, which is usually solved by heuristic methods that sacrifice optimality for efficiency. How to improve the quality of heuristic scheduling algorithms is always an importance topic. Random optimization is a powerful tool to solve many NP-hard problems. However, it has not been widely used in scheduling due to the various, complex constraints imposed during scheduling. Chapter 3 presents a new way of improving schedule quality by using random walks to explore the solution space. To improve the efficiency of the solution space exploration, two techniques are proposed: 1) utilize the feedback of the max-flow algorithm for next perturbation; 2) keep the dependent constraint consistent between operations. Experiment shows that under the same resource usage, the enhanced max-flow scheduling algorithm reduces schedule latency by up to 10%. This result testifies the ability of the algorithm to discover more circuit optimization opportunities than conventional heuristics.

Chapter 4 [Synthesis of General Floating-Point Arithmetic Units (FPU)] works on another essential task in HLS – support of floating-point arithmetic. Currently, the most popular implementations of floating-point functions are based on piecewise polynomial approximation, which partitions domain $X$ of function $f(x)$ to segments and approximates $f(x)$ by a polynomial in each segment. However, existing implementations cannot cope with a large number of segments and therefore only support special functions of narrow domain. To synthesize more general functions of wide domain, efficient handling of excessive segments is needed. Chapter 4 proposes the aligned partition of domain $X$ that reduces the cost of processing excessive (up to 2,000) segments, making functions of wider domain synthesizable. Chapter 4 also presents two hardware architectures for the evaluating polynomial: 1) type one is a fully pipelined architecture that has high throughput, fair accuracy, low delay, and moderate resource usage; 2) type two is a compact architecture that has small size, high accuracy, fair delay, and moderate throughput. Experiment shows that the proposed method is capable of synthesizing a much wider range of floating-point functions than previous methods. On hyperbolic functions, the result even outperforms the fixed-point manual implementations (20k times smaller error rate, 50% reduction of LUT and register usage). The proposed FPU synthesis method introduces design automation into a domain that traditionally requires intensive manual works. It has very practical usage in HLS.

Chapter 5 [Conclusion and Future Works] concludes the whole research works of
this thesis on scheduling algorithm and floating-point function synthesis. The directions for future works are discussed as well.