Bitwidth-Aware Scheduling and Binding in High-Level Synthesis

Jason Cong[+], Yiping Fan[+], Guoling Han[+], Yizhou Lin[+], Junjuan Xu[*+], Zhiru Zhang[+], Xu Cheng[*]
[+]Computer Science Department, UCLA, Los Angeles CA 90095 USA
[*]Computer Science and Technology Department, Peking University, Beijing 100871 PRC

Abstract - Many high-level description languages, such as C/C++ or Java, lack the capability to specify the bitwidth information for variables and operations. Synthesis from these specifications without bitwidth analysis may introduce wasted resources. Furthermore, conventional high-level synthesis techniques usually focus on uniform-width resources, thus they cannot obtain the full resource savings even with bitwidth information. This work develops a bitwidth-aware synthesis flow, including bitwidth analysis, scheduling and binding, and register allocation and binding, to exploit the multi-bitwidth nature of operations and variables for area-efficient designs. We also develop lower bound estimation to evaluate the efficiency of our proposed solutions for register allocation and binding. The flow is implemented in the MCAS synthesis system [11]. Experimental results show that our proposed bitwidth-aware synthesis flow reduces area by 36% and wire-length by 52% on average compared to the uniform-width MCAS flow, while achieving the same performance.

I. Introduction

The gap between design productivity and complexity continues to grow larger. According to [4], the increasing rates for VLSI complexity and design productivity are 58% and 21% per year, respectively. This large gap must be shortened to satisfy the time-to-market and design cost requirements. Using high-level languages is one of the most promising solutions for improving design productivity by raising the level of abstraction. Alleviating the design complexity and producing high-quality products are the two key points for making high-level languages successful and accepted by designers.

Recent system-level languages from academia or industry, such as SystemC [3] and SpecC [2], can specify variables and operations with arbitrary bitwidths. However, traditional high-level languages such as C/C++ only provide computation types with restricted lengths, e.g., an 8/16/32/64 bitwidth. Experimental results show that there are 40% redundant bits on average in a set of benchmark programs written in C [21]. Applying the bitwidth analysis technique proposed in [21] for nine real-life benchmarks from [20], we obtained similar results showing that there is a 36% redundancy for operations and 21% for variables.

Potential hardware resource cost can be reduced if these redundant bits are identified. Empirical relations between area usage and bitwidth for multipliers and adders can be found in [12]. An adder's area is proportional to its input bitwidth, and a multiplier's area is proportional to the product of its two input bitwidths. In practice, we obtained similar results in Altera's Stratix FPGA devices: a 16-bit adder uses 17 logic elements, and a 32-bit adder uses 35; for 16×16- and 32×32-bit multipliers, the logic element usages are 380 and 1325 respectively. For other resource types such as registers and multiplexers, we can reasonably assume a linear relationship between area and bitwidth. Wiring cost, which is even more expensive in nanometer technologies, will also be saved, thanks to the reduction of the operation bits.

In RTL languages, explicitly declared bitwidths are used for variables, registers, functional units (FU), and data buses to reduce area and power. However, specifying bitwidths explicitly for hardware designs starting from a traditional algorithmic description is time consuming. An ideal design flow is the one that can automatically analyze bitwidths for variables and operations to alleviate the designer's burden and reduce the opportunity for errors.

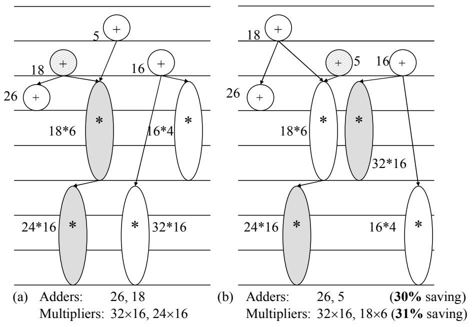
5
+
18
16
26
18*6
*
16*4
24*16
32*16
(a) Adders: 26, 18
Multipliers: 32×16, 24×16
18
5
16
26
18*6
32*16
24*16
16*4
(b) Adders: 26, 5 (**30%** saving)
Multipliers: 32×16, 18×6 (**31%** saving)

Figure 1. Two scheduling and binding solutions of multi-bitwidth operations.

Furthermore, even after the precise bitwidth information is available, most conventional synthesis methods focus only on the uniform bitwidth resources, and cannot take full advantage of the bitwidth information. To obtain good quality-of-result, awareness of the bitwidth information should be added into the high-level synthesis flow. We will use an example to illustrate the inefficiency of the traditional approach. Figure 1 shows two scheduling and FU binding solutions for a same data flow graph (DFG). Nodes represent operations (additions or multiplications), and edges represent data dependence. The required minimum operation bitwidth is annotated on each node. The required latency is 8 cycles, and the execution times for adders and multipliers are 1 and 3 clock cycles, respectively. Note that operations bound to the same FU are colored in the same way. For each solution, two adders and two multipliers are required. However, FUs for (b) cost less area due to smaller bitwidths, 30% saving for adders and 31% for multipliers. From this example, we learn that scheduling and binding without considering the bitwidth information may produce sub-optimal results. Because of the great impact on area, and thus on power, connection, and clock period, optimization by exploiting bitwidths of variables and operations becomes one important issue in high-level synthesis.

Recently, several techniques have been proposed for bitwidth optimization in architecture and high-level synthesis. Constantinides [12] formulated the scheduling and FU binding problem as an ILP, and proposed an iterative solution in [13]. However, the proposed binding solution, which selects the clique with the maximum ratio of clique size and clique cost, is not suitable for register binding. The work in [17] uses the force-directed scheduling to balance the number of bits per cycle for additive operations. One limitation is that this method can only be applied to additive operations. In addition, the resulting controller may have unaffordable cost due to the large number of small adders. The work in [5] proposes an iterative scheduling method for a data flow graph with bitwidth information. Complete allocation and binding of FUs and registers are performed to obtain cost for each feasible scheduling. Because of the large search space, the scheduling may be extremely slow. There are other techniques proposed to optimize a datapath using bitwidth information from the architecture point of view [6] [9]. However, none of the previous works on high-level synthesis consider interconnect delay for bitwidth-aware scheduling, or study the multiple bitwidth register allocation and binding thoroughly.

In this paper we propose a complete bitwidth-aware high-level synthesis flow from a behavioral C description to a RTL VHDL implementation based on the MCAS synthesis system [11]. MCAS is an architectural synthesis system on top of a recently proposed Regular Distributed Register (RDR) micro-architecture [10]. In this paper, our goal is to minimize the total bitwidth for FUs and registers while maintaining the performance. Specifically, we propose a simultaneous scheduling and binding to minimize the total bitwidth of FUs with consideration of interconnect delay. For register allocation and binding, we transform it into a minimum weighted-interval-graph coloring problem and propose an efficient heuristic solution. The results approach the lower bound with only a 0.05% gap presented. Experimental results show that our synthesis flow can achieve a significant amount of savings in terms of area and wire-length.

We will present the overall bitwidth-aware synthesis flow and define the problem to be solved in Section II. Section III introduces the simultaneous scheduling and binding algorithms. Section IV presents the detailed lower-bound estimation and heuristic solution for the register allocation and binding. Section V shows experimental results and Section VI concludes this paper.

II. Bitwidth-Aware Synthesis Flow

In Figure 2 we illustrate the proposed bitwidth-aware high-level synthesis flow, which is composed of four steps. First, a behavioral description in C is transformed into the Machine-SUIF [19] intermediate representation. After compilation optimizations, such as dead code elimination and peep-hole optimization are applied, the bitwidth analysis is performed as a stand-alone Machine-SUIF pass. The goal of the bitwidth analysis is to automatically decide the minimum bitwidth for each variable and operation while retaining the program correctness. We adopt the bitwidth analysis method introduced in [21] which uses constants, array indices and computation to decide the minimum bitwidth. The output is a DFG annotated with bitwidth information.

In the second step, the MCAS architectural synthesis system is utilized to perform scheduling, binding and placement. MCAS is a synthesis system targeting RDR micro-architecture, which is an integrated architectural and synthesis solution for multi-cycle on-chip communication. A RDR chip is divided into an array of islands. The size of each island is chosen such that all intra-island communications can be performed in a single clock cycle, and the inter-island communications take multiple cycles. Its regularity facilitates the predictability of interconnect delays at early design stages. The output of this step is a scheduled and bound DFG with placement information. In this step, clock period and clock cycles are traded off to optimize speed performance, while bitwidth information is not explored.

In the third step, bitwidth-aware re-scheduling and re-binding is performed to minimize area cost of FUs with consideration of interconnect delay obtained from placement information. And bitwidth-aware register allocation and binding is performed to minimize area cost of registers. After the corresponding datapath and controller are generated in the last step, the whole RTL design is written into the output VHDL files.

In the following sections we will present the bitwidth-aware high-level synthesis of the third step in detail. The **multiple bitwidth scheduling and binding problem** is formulated as follows:

Given: (1) A DFG annotated with bitwidths, (2) a time constraint, (3) placement information of functional units, and (4) a resource IP library, where each resource type has arbitrary bitwidth configurations, each of which is associated with an area cost.

Objective: Schedule and bind the DFG into the library with consideration of interconnect delay from placement and without violating the time constraint, such that the final area of the required resources is minimized.

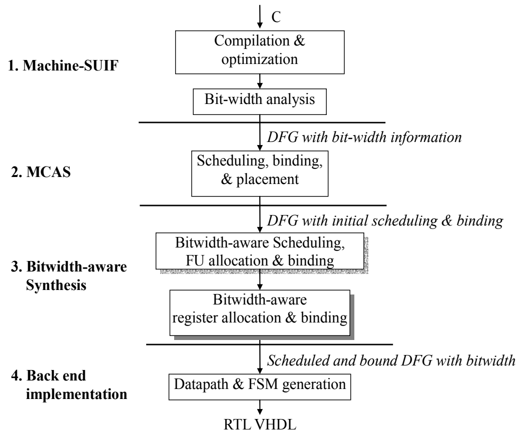
C
Compilation & optimization
1. Machine-SUIF
Bit-width analysis
DFG with bit-width information
2. MCAS
Scheduling, binding, & placement
DFG with initial scheduling & binding
Bitwidth-aware Scheduling, FU allocation & binding
3. Bitwidth-aware Synthesis
Bitwidth-aware register allocation & binding
Scheduled and bound DFG with bitwidth
4. Back end implementation
Datapath & FSM generation
RTL VHDL

Figure 2. The proposed bitwidth-aware synthesis flow.

With a target design frequency, the time constraint becomes a number constraint of clock cycles (or control steps), and the execution time of operations is represented by cycle numbers. In this work, we set the target clock period as 5ns in MCAS system. We assume a unified latency for a functional unit type, in particular, 1 cycle for adders and 3 cycles for multipliers. To simplify the presentation of the problems, we assume all the IPs are non-pipelined, and IPs with the same type have a same latency regardless of bitwidth configurations. However, the approaches we proposed in this paper can be easily extended to handle pipelined IPs or IPs with various latencies for different bitwidth configurations.

Since the area of an IP is a function of its bitwidth (Section I), in this paper we focus on minimizing the total bitwidth for each kind of resource, which can directly lead to minimized area. We solve the multi-bitwidth scheduling and binding problem in two phases. First, we try to schedule and bind the operations to minimize FUs. Register allocation and binding are then performed on the scheduled DFG to minimize the total bitwidth of required registers. Since the bitwidth for FUs and registers is reduced, the wirelength is optimized accordingly.

III. Scheduling and Binding

In this section we first introduce the lower-bound estimation of FU bitwidth for a DFG. The lower-bound-based simultaneous scheduling and binding are then presented. Through the scheduling and binding sub-task, we will consider the interconnect delay obtained from placement information given by MCAS.

A. Preliminaries

In a DFG after bitwidth analysis, each operation is annotated with the information of required input/output bitwidths. The input-width constraint indicates a bitwidth requirement for a FU to perform the operation correctly, and the output-width constraint enforces a register with enough bitwidth to hold the result value.

Only two types of operations, addition and multiplication (*ADD* and *MUL*, for short), are considered in this work. We assume each operation o has a single output with bitwidth b_0, and two inputs with bitwidth pair (b_1, b_2). It is clear that the bitwidth of a variable produced by operation o is b_0.

A multiplier can be configured as $b_1 \neq b_2$, but the wider input dominates the cost of the implementation. For an adder, a general interface restriction of a conventional IP library requires that the inputs must have the same bitwidths. Therefore, we only consider the larger bitwidth as the bitwidth of an operation for both multiplication and addition, and define *the bitwidth of operation o* as $b(o) = max\{b_1, b_2\}$, where o has input bitwidth pair (b_1, b_2).

B. Lower-Bound Estimation of FUs

Previous research on the FU lower-bound estimation only focuses on the number of FUs [7] [8] [14] [18] [22]. In this section we will introduce a lower-bound estimation for the total bitwidth of FUs with various bitwidth. In particular, our solution extends the technique of [18] to support multi-bitwidth FUs. The bitwidth lower bound is the maximum of the minimum resource requirement for each interval.

The minimum resource requirement for an interval is calculated in two steps. First, $ASAP(o)$ and $ALAP(o)$, the as-soon-as-possible and as-late-as-possible control steps of operation o are computed for a DFG under time constraint T. If operation o is unscheduled, $ASAP(o)$ and $ALAP(o)$ correspond to the earliest and latest feasible control step, respectively. Otherwise, $ASAP(o)$ and $ALAP(o)$ are both equal to the scheduled control step. For a time interval $[p, q] \subseteq [1, T]$, *the minimum overlap* between operation o and the interval is denoted as $\lambda(o, p, q)$ and calculated as follows:

$\lambda(o, p, q) = min\{ | [ASAP(o), ASAP(o)+exe(o)] \cap [p, q] |,$
$| [ALAP(o), ALAP(o)+exe(o)] \cap [p, q] | \},$

where $exe(o)$ stands for the required execution time of o.

After the calculation of overlap numbers, we insert $\lambda(o, p, q)$ copies of $b(o)$ into the *bitwidth list* of $[p, q]$, denoted by $L_f(p, q)$, where f is the type of operation o, either *ADD* or *MUL*.

Figure 3 shows the ASAP and ALAP schedules for Figure 1 under the time constraint $T = 8$. The overlap between the multiplications, a, b, c and d, and interval $[4, 7]$ can be calculated as

$$\lambda(a_{18*6}, 4, 7) = 1,\ \lambda(b_{24*16}, 4, 7) = 2,$$
$$\lambda(c_{32*16}, 4, 7) = 1,\ \lambda(d_{16*4}, 4, 7) = 1.$$

Based on the results, the bitwidth list of interval $[4, 7]$ for the multiplications is $L_{MUL}(4, 7) = \{18, 24, 24, 32, 16\}$.

In the second step, the required minimum FU bitwidth is calculated for each interval. For a constructed bitwidth list $L_f(p, q)$ of interval $[p, q]$ for operation type f, the bitwidths are sorted in non-increasing order. It is easy to understand that the required minimum number of the FUs of type f during this interval is $n_f(p, q) = \lceil |L_f(p, q)| / (q-p+1) \rceil$.

The sorted list $L_f(p, q)$ is then partitioned into $n_f(p, q)$ sub-lists of size q-p+1 each, except that the last sub-list might be smaller. We choose the largest one (i.e, the first one) from these sub-lists to form a *dominant bitwidth list* $D_f(p, q)$, which contains the required bitwidths of FUs to execute the operations during interval $[p, q]$.

For the example in Figure 3, $L_{MUL}(4, 7)$ will be partitioned to sub-lists $\{32, 24, 24, 28\}$ and $\{16\}$. The dominant bitwidth list $D_{MUL}(4, 7) = \{32, 16\}$ indicates that the DFG in Figure 3 needs at least two multipliers of bitwidths 32 and 16.

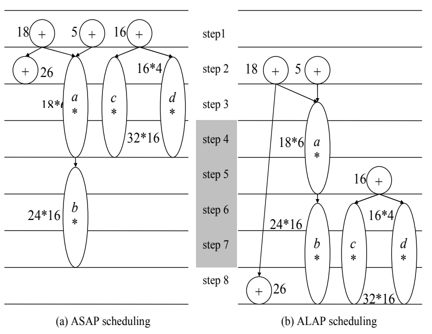

Figure 3. An illustration of the lower-bound calculation.

Theorem 1: With the dominant bitwidth lists for all intervals within $[1, T]$, the *lower bound of the required bitwidths for each instance of operation type f* is

$$\zeta_f = \left\{ \max_{1 \le p \le q \le T} \left\{ D_f(p, q)^{(k)} \right\} \right\},$$

where $D_f(p, q)^{(k)}$ denotes the kth element of $D_f(p, q)$, and $D_f(p, q)^{(k)} = 0$ if $k > |D_f(p, q)|$. □

This formula is based on the fact that the maximum of the kth bitwidth of all dominant bitwidth lists is the minimum required bitwidth for the kth instance of FUs of type f. The time complexity will be $O(T^2(n+nlog(n)))$.

After the bitwidth lower bound is computed for each instance of all operation types, we have *weighted-area lower-bound* Φ of a DFG as follows:

$$\phi = \sum_{1 \le i \le |\zeta_{add}|} \zeta_{add}^{(i)} + \alpha \sum_{1 \le i \le |\zeta_{mul}|} \zeta_{mul}^{(i)} \cdot \zeta_{mul}^{(i)}$$

where α is a ratio weight of multiplier area over adder area, and $\zeta_f^{(i)}$ is the ith element of ζ_f. Due to the fact that the area of a multiplier is proportional to the product of its two input bitwidths (Section I), we use the square of bitwidth to represent the area cost for multipliers. Since the minimum overlaps, bitwidth lists, dominant bitwidth lists, and ζ_f are all decided by the current scheduling status S of the DFG, the weighted-area lower bound is also a function of S, denoted as $\Phi(S)$. We will use this weighted-area lower bound technique in the following scheduling algorithm.

C. Scheduling and Binding Algorithm

Based on the lower bound computation stated above, simultaneous scheduling and binding are performed to minimize the total bitwidth of required FUs with consideration of interconnect delay and without violating time constraint.

The basic idea is to schedule an operation at a control step such that the resulted weighted-area lower bound is kept as small as possible. In each step, suppose S is the current scheduling status, recording the control steps for scheduled operations and feasible control steps for un-scheduled operations, and S' is the updated scheduling status of S if an unscheduled operation o had been scheduled to control step c, we define

$$LB(S, o, c) = \Phi(S').$$

This metric indicates the impact of the scheduling step on the potential resource requirements, which is estimated as the weighted-area lower bound. We first choose a pair (o, c) with minimum $LB(S, o, c)$. FU binding is then performed to decide whether operation o can be scheduled at step c finally. If there is an available FU f at step c, o will be scheduled and bound. Otherwise, the next pair of (o, c) with minimum $LB(S, o, c)$ among the remaining operations and steps will be considered. The checking rule for binding o to f is that data dependence is maintained between o and its scheduled and bound predecessors and successors with consideration of interconnect delay. Specifically, for a predecessor p of o, which is scheduled at c_p and bound to f_p, the following expression needs to hold true to keep the data dependence:

$$c_p+exe(p)+delay(f, f_p) \leq c,$$

where $delay(f, f_p)$ is the interconnect delay between FU f and f_p obtained from the placement information as determined by MCAS (step 2 in our flow). Similar expressions are checked for successors. The iterative choosing process will guide the algorithm to minimize the area of required resources.

In the algorithm implementation, we apply two methods to reduce the lower-bound updating time and operation selecting time without sacrificing solution quality. The first is to update bitwidth lists incrementally, which is the base for lower-bound calculation and is in the inner loop of our scheduling algorithm.

Theorem 2: Given a DFG and a scheduling status S. After one or more unscheduled operations are scheduled and the scheduling status becomes S', the new minimum overlap number $\lambda'(o, p, q)$ under S' is no less than $\lambda(o, p, q)$ under S, for any operation o and interval $[p, q]$.

Proof: When o is scheduled to any control step between $ASAP(o)$ and $ALAP(o)$, the overlap number with $[p, q]$ is no less than that when o is scheduled to $ASAP(o)$ or $ALAP(o)$ [18]. After the scheduling status is updated, regardless of whether o is scheduled or not, the new feasible lifetime $[ASAP'(o), ALAP'(o)]$ is a subset of the old one $[ASAP(o), ALAP(o)]$. Combining the previous definition of minimum overlap, we have the conclusion that $\lambda'(o, p, q) \geq \lambda(o, p, q)$. □

According to *Theorem 2*, each time the scheduling status is updated, the bitwidth list $L'_f(p, q)$ can be obtained by adding $\lambda'(o, p, q)-\lambda(o, p, q)$ copies of $b(o)$ into $L_f(p, q)$. Since the feasible lifetime may not change for some operations, only those operations whose feasible lifetimes are changed need to be checked to update $L_f(p, q)$. Furthermore, only those bitwidth lists which are indeed updated need to be checked to recalculate the new weighted-area lower bound Φ.

Theorem 3: Given a DFG and a scheduling status S. After one or more unscheduled operations are scheduled and the scheduling status becomes S', the new weighted-area lower bound $\Phi(S')$ is no less than $\Phi(S)$.

Proof: After one or more operations are scheduled, the new overlap result $L'_f(p, q)$ is a superset of $L_f(p, q)$, which is inferred by *Theorem 2*. From the computation of Φ shown in the previous section, it is straightforward that the new lower bound $\Phi(S')$ is no less than $\Phi(S)$. □

Based on *Theorem 3*, since $\Phi(S')$ is no less than $\Phi(S)$, $\Phi(S)$ is the lower bound of $LB(S, o, c)$ over all o and c. When one $LB(S, o, c)$ is equal to $\Phi(S)$, o must be one of the operations that have the least value of LB. We can skip the computation of LB for the remaining unscheduled operations, schedule o to control step c, and enter the next scheduling iteration.

IV. Register Allocation and Binding

In a traditional high-level synthesis flow, register allocation aims to decide the required registers, and binding is defined as the explicit mapping from variables to register instances. For variables with multiple bitwidths, it is natural to combine the allocation and binding together to improve the solution quality. We propose a weighted-interval-graph coloring algorithm to cope with these two tasks simultaneously. In our synthesis flow, we perform register allocation and binding after the simultaneous scheduling and binding. We assume that optimizations for other resources, such as ports and multiplexers, can be performed afterward.

A. *Problem Definition*

For a scheduled DFG with uniform bitwidths, the problem of minimizing the number of registers can be reduced to the chromatic number problem in an interval graph, which can be solved with the left-edge algorithm in polynomial time [16]. In the remainder of this paper, we denote the minimum chromatic number of graph G as $\chi(G)$. Considering registers with various bitwidths, the new problem is to minimize the total bitwidth of registers.

Some definitions and concepts related to this problem are introduced below. Let $s(a)$ denote the control step where variable a is generated, and $t(a)$ denote the last control step where a is consumed by other operations. Given a scheduled DFG with bitwidth information, a ***weighted interval graph*** $G(V, E)$ can be derived, where each vertex in V corresponds to a variable in the DFG and edge $(a_1, a_2) \in E$ if and only if the lifetimes of the two corresponding variables a_1 and a_2 have non-trivial overlap, i.e.,

$$[s(a_1), t(a_1)] \cap [s(a_2), t(a_2)] \neq \varnothing.$$

It is clear that two variables can be bound to a same register only if their lifetimes do not overlap. Therefore, a coloring scheme of G corresponds to a binding solution for variables. All variables with the same color are bound to the same register.

Let $w(v)$ denote the *weight of vertex v*, $w(v) = b(a)$, where a is the corresponding variable of v. We define *the weight of a vertex set V* as $W(V) = max\{w(v) \mid v \in V\}$, and *the weight of a graph* $G(V, E)$ as $W(G) = W(V)$. Suppose that a proper coloring scheme $P = \{c_1, c_2, \ldots, c_k\}$ of $G(V, E)$ partitions V into vertex subsets $V_1, V_2, \ldots,$ and V_k, *the weight of color* c_i, for $1 \leq i \leq k$, is defined as $W(V_i)$, and *the weight of the coloring scheme P* is defined as

$$\Psi(G, P) = \sum_{1 \leq i \leq k} W(V_i).$$

From the above definitions, the bitwidth-aware register allocation and binding problem is equivalent to a **weighted-interval-graph coloring problem** as follows:

Given*:* A weighted interval graph $G(V, E)$.

Objective*:* Find a coloring scheme P of G, such that the weight of the coloring scheme P, $\Psi(G, P)$ is minimized.

When the weights of vertices are uniform, this problem can be solved in polynomial time [16]. However, the complexity of the general problem with various weights remains unknown. In the following sections, a lower-bound estimation for $\Psi(G, P)$ and a heuristic solution will be presented. Experimental results show that only a 0.05% gap is presented between the solution and lower bound.

B. *Lower-Bound Estimation of Registers*

We will use a register-sharing problem to illustrate how to calculate the lower bound of the weight of the coloring schemes for a weighted interval graph. Figure 4(a) shows the lifetimes of five scheduled variables, with bitwidths annotated, and Figure 4(b) is the derived weighted interval graph $G(V, E)$. We first divide G

into subgraphs according to the weights of the vertices. All vertices in each subgraph have the same weight. It is obvious that these subgraphs and their unions still keep the interval property. We then sort these subgraphs in the decreasing order of weights.

For the example in Figure 4, the sorted subgraphs will be $G_1 = \{a, b\}$, $G_2 = \{c, e, f\}$, and $G_3 = \{d\}$. From G_1, we know that the number of the colors with weight *28* must be no less than $\chi(G_1) = 1$ for any coloring scheme of G. Then for $G_1 \cup G_2$, the number of colors with *weight no less than 16* must be at least $\chi(G_1 \cup G_2)$. Combing these two facts, we conclude that the number of colors with *weight 16* must be at least $\chi(G_1 \cup G_2) - \chi(G_1) = 1$. Similarly for $G_1 \cup G_2 \cup G_3$, the color number with weight *5* must be no less than $\chi(G_1 \cup G_2 \cup G_3) - \chi(G_1 \cup G_2) = 1$. Basing on these results, we conclude that the lower bound of the weight of any coloring scheme for this interval graph will be $28 \times 1 + 16 \times 1 + 5 \times 1 = 49$.

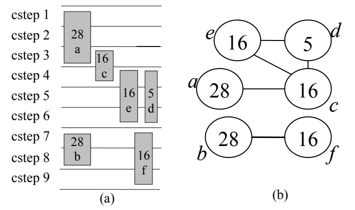

Figure 4. (a) Lifetimes and (b) the derived interval graph.

Theorem 4: For any proper coloring scheme P for a weighted interval graph G, $\Psi(G, P) \geq L(G)$, *where*

$$L(G) = \sum_{1 \leq i \leq m} \left[W(G_i) \cdot \left(\chi\left(\bigcup_{1 \leq j \leq i} G_j \right) - \chi\left(\bigcup_{1 \leq j \leq i-1} G_j \right) \right) \right],$$

where G_i, for $1 \leq i \leq m$, are the sorted subgraphs constructed in the aforementioned way.

Proof: This is straightforward and correct given the definitions of $\Psi(G, P)$ and $L(G)$. The formal proof is omitted here. □

C. Allocation and Binding Algorithm

We use the ideas in the lower-bound calculation to solve the weighted-interval-graph coloring problem. Again, we use the same example in Figure 4(b) to illustrate the heuristic solution.

After the weight-based partitioning and sorting of G into subgraphs $G_1 = \{a, b\}$, $G_2 = \{c, e, f\}$, and $G_3 = \{d\}$, we first try to move vertices from G_i, for $2 \leq i \leq 3$, to G_1 to decrease the chromatic number for each G_i without increasing $\chi(G_1)$. Since G_2 has the largest weight among the three subgraphs, reducing $\chi(G_2)$ is preferred. Noting that the chromatic number χ is equal to the maximum clique size for an interval graph, we always try to move the vertices from the maximal cliques of G_i to G_1. In this way, $\chi(G_i)$ can be directly decreased. For G_2, the only maximum clique is $\{c, e\}$. Moving c from G_2 to G_1 decreases $\chi(G_2)$ by 1, but increase $\chi(G_1)$ at the same time. However, moving e can safely decreases $\chi(G_2)$ without increasing $\chi(G_1)$. Therefore, e is chosen to add into G_1 finally. Since no vertex in the new G'_2, $\{c, f\}$, can be safely moved into G_1, we continue to process G_3. Adding d into G_1 increases $\chi(G_1)$, so no moving happens for G_3. Now, processing on G_1 is done. We then move on to process G_i, for $2 \leq i \leq 3$ in the same way.

The final partition of G is $G_1 = \{a, b, e\}$, $G_2 = \{c, f\}$, and $G_3 = \{d\}$, and the coloring scheme will be $P = \{\{a, b, e\}, \{c, f\}, \{d\}\}$ with weight *49*, which is equal to the lower bound calculated in the previous section.

Figure 5 presents the pseudo code of the heuristic algorithm, weighted interval-graph coloring (IGC). The formal algorithm description is as follows. During the processing of subgraph G_i, we try to move vertices from G_j, for $i+1 \leq j \leq m$, to G_i. The moving process consists of two phases, both of which process the subgraphs in the decreasing order of their weights. The first phase selects one vertex from each maximum clique of G_j to compose a vertex set S and merge it to G_i, such that the chromatic number of G_i will not increase. Since the cardinality of the maximum clique equals the chromatic number for interval graphs, decreasing the cardinality of all the maximum cliques in G_j will directly reduce the chromatic number. Therefore, taking vertex set S out from G_j decreases $\chi(G_j)$ by *1*.

When no operations from G_j, for $i+1 \leq j \leq m$, can be moved to G_i in the first phase, the second moving phase is performed. Now we randomly select a vertex from G_j to move into G_i without increasing $\chi(G_i)$. The purpose of this phase is to maximize the utilization of colors with high weight, and thus provide further chances for reducing the number of colors with lower weight.

After the operation moving is done, the conventional left-edge or other optimization techniques, such as connection and MUX minimization algorithms, can be separately applied for each new subgraph G'_1, $G'_2, \ldots, G'_{m'}$, where $m' \leq m$. The weight for the coloring scheming is $\Sigma_{1 \leq i \leq m'} W(G'_i) \cdot \chi(G'_i)$.

```
Procedure Weighted_Interval_Graph_Coloring
Input:
    Weighted interval graph G
Output:
    Coloring scheme of G

Partition and sort G into {G_1, G_2,..., G_m}, with the decreasing order of the weights
for each G_i, 1 ≤ i ≤ m-1
        Compute χ(G_i)
        for each G_j, i+1 ≤ j ≤ m
            while ∃S ⊆ G_j, such that χ(G_j-S) = χ(G_j)-1 and χ(G_i∪S) = χ(G_i)
                G_i ← G_i ∪ S
                G_j ← G_j - S
        for each G_j, i+1 ≤ j ≤ m
            while ∃v ∈ G_j , such that χ(G_i∪{v}) = χ(G_i)
                G_i ← G_i ∪ {v}
                G_j ← G_j - {v}
Coloring each new subgraph separately
```

Figure 5. Algorithm for weighted-interval-graph coloring.

In RDR, registers are located in each island independently. After register allocation and binding are done for every island, extra registers are inserted between different islands to achieve multi-cycle communication. Due to page limitation, we will not introduce RDR in detail.

V. Experimental Results

A. Register Allocation and Binding

Table 1. Bitwidth-aware register binding.

Designs	LB	Left-Edge	KS[15]	Weighted IGC
aircraft	1270	1402	1335	1270
chem	896	962	929	897
dir	474	487	505	474
honda	312	328	368	313
lee	216	216	232	216
mcm	689	721	691	689
pr	270	297	298	270
u5ml	1717	1892	1778	1717
wang	269	293	302	269
Ave gap	1	+6.6%	+7.5%	+0.05%

To evaluate the efficiency of the heuristic algorithm for coloring, we conducted experiments on a set of real-life benchmarks from [20], including several different DCT algorithms such as *pr*, *wang*, *lee*, and *dir*, and several DSP programs such as *aircraft*, *mcm*, *honda*, *chem*, and *u5ml*. The results, represented by the total bitwidth, are summarized in Table 1. The second column presents the lower-bound results from Section IV.B. The third column is the result from left-edge plus bitwidth post-processing, which is to set the bitwidth of a register as the maximum bitwidth of all variables stored in it. The fourth column presents the results from the clique-partition method of [15]. The last column is the result given by our proposed solution. The three methods share the same scheduling method to generate the scheduled DFGs. The left-edge algorithm can produce the minimum register number, but it does not consider the bitwidths of variables while doing binding. KS noticed the bitwidth influence but it suffers from the larger number of registers. The results of left-edge and KS are 6.6% and 7.5% higher than the lower bound, while our method is only 0.05% higher. This shows that our lower-bound estimation and heuristic solution for the weighted-interval-graph coloring problem are both close to optimal solutions.

B. *Comparison of Three Flows*

Our synthesis flow is implemented in a C++/UNIX environment. In order to obtain the final performance results, Altera's Quartus II version 2.2 0 is used to synthesize the resulting RTL VHDL onto the FPGA device Stratix^{TM} EP1S80F1508C6. We use the default compilation options in Quartus II.

To validate our proposed bitwidth-aware synthesis flow, we set up three experimental synthesis flows as follows. All of them have the same control step constraint, and share the same backend to generate datapath and controllers.

- *Flow1(MCAS)*: MCAS generates the scheduling and binding results and placement information. All operations and variables have uniform bitwidth (32-bits).
- *Flow2(MCAS+MB-PP)*: Perform a bitwidth post-processing after *Flow1* is done, which is to set the bitwidth of a FU as the maximum bitwidth of all operations executed on it, and set the bitwidth of a register as the maximum bitwidth of all variables stored in it.
- *Flow3(MCAS-MB)*: After MCAS generates the scheduling and binding results and placement, the lower-bound-based scheduling & binding and the bitwidth-aware register allocation and binding are performed.

Table 2. FPGA compilation results of three synthesis flows.

Design	Node#	*MCAS*		*MCAS+MB-PP*		*MCAS-MB*	
		LE	WL(k)	LE	WL(k)	LE	WL(k)
aircraft	422	-	-	10559	267	6860	181
chem.	342	8339	247	7101	191	4814	136
dir	127	2810	91	2075	48	1135	27
honda	107	2433	77	1774	38	1124	24
lee	49	1033	54	722	35	614	25
mcm	94	2562	105	2411	83	2392	75
pr	42	1194	63	1030	45	967	38
u5ml	565	14447	396	12774	318	7143	166
wang	48	1275	73	1078	36	1050	38
Ave	-	1	1	-18.1%	-34.5%	-36.3%	-51.5%

Table 2 shows the experimental results for these three flows, including area results for datapath and control logic in terms of logic element (LE) and wire-length (WL) reported by Quartus II. The column "*Node#*" lists the node number for each benchmark. *Flow3* reduces area and wire-length by 36.3% and 51.5% respectively compared to *Flow1* when averaged over the benchmarks (excluding aircraft). When compared to *Flow2*, *Flow3* still reduces area and interconnect by 24.2% and 26.4%, respectively.

Design *aircraft* fails to fit into the selected FPGA device in *Flow1* because of the large number of its I/O pins. Interestingly, with the same bitwidth constraints for the primary inputs, bitwidth analysis reduces the bitwidth requirements for the primary outputs so that the I/O fitting problem is resolved for the other two flows.

VI. Conclusions

We have presented a complete bitwidth-aware high-level synthesis flow based on MCAS synthesis system, including bitwidth analysis, simultaneous scheduling and binding, and a weighted-interval-graph coloring solution for register allocation and binding. Lower-bound calculation is used both for estimation of potential improvement of the existing solutions and for development of the heuristic algorithms. Experimental results show that our bitwidth-aware synthesis flow achieves significant reduction for area (36%) and wire-length (52%) for a set of benchmark designs.

Acknowledgements

This research is funded by National Science Foundation under award CCR-0096383.

References

[1] Altera Web Site, http://www.altera.com.
[2] SpecC Web Site, http://www.ics.uci.edu/~specc.
[3] SystemC Web Site, http://www.systemc.org.
[4] "The National Technology Roadmap for Semiconductors: Technology Needs," Semiconductor Industry Association, 1997.
[5] V. Agrawal, A. Pande, and M. M. Mehendale, "High Level Synthesis of Multi-Precision Data Flow Graphs," *Proc. of the 14th International Conference on VLSI Design*, 2001.
[6] Y. Cao and H. Yasuura, "A System-level Energy Minimization Approach Using Datapath Width Optimization," *Proc. of ISLPED*, 2001.
[7] S. Chaudhuri and R. A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," *Proc. of the 7th International Symposium on High-Level Synthesis*, 1994.
[8] S. Chaudhuri, R. A. Walker and J. E. Mitchell, "Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem," *IEEE Trans. on VLSI Systems,* 1994.
[9] J. Choi, J. Jeon and K. Choi, "Power Minimization of Functional Units by Partially Guarded Computation," *Proc. of ISLPED*, 2000.
[10] J. Cong, Y. Fan, X. Yang, and Z. Zhang, "Architecture and Synthesis for Multi-Cycle Communication," *Proc. of ISPD*, 2003.
[11] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and Synthesis for On-Chip Multicycle Communication," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* 2004.
[12] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimal Datapath Allocation for Multiple-Wordlength Systems," *IEEE Electronics Letters*, 2000.
[13] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Heuristic Datapath Allocation for Multiple Wordlength Systems," *Proc. of Design, Automation and Test in Europe*, 2001.
[14] E. B. Fernandez and B. Bussell, "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedule," *IEEE Trans. on Computers*, 1973.
[15] K. Kum and W. Sung, "Combined Word-Length Optimization and High-Level Synthesis of Digital Signal Processing Systems," *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 2001.
[16] G. D. Micheli and G. Demicheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Inc., 1994.
[17] M. C. Molina, J. M. Mendias, and R. Hermida, "High-Level Synthesis of Multiple-Precision Circuits Independent of Data-Objects Length," *Proc. of Design Automation Conference*, 2002.
[18] A. Sharma and R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applications," *IEEE Trans. on VLSI Systems*, 1993.
[19] M. D. Smith and G. Holloway, "An Introduction to Machine SUIF and its Portable Libraries for Analysis and Optimization," Division of Engineering and Applied Sciences, Harvard University.
[20] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *IEEE Trans. on VLSI Systems*, 1995.
[21] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth Analysis with Application to Silicon Compilation," *Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2000.
[22] M. Narasimhan, and J. Ramanujam, "On lower bounds for scheduling problems in high-level synthesis," *Proc. of Design Automation Conference*, 2000