A Unified Lower Bound Estimation Technique for High-Level Synthesis [1]

Seong Yong Ohm §
Fadi J. Kurdahi †(Contact Author)
Nikil Dutt ‡

April 3, 1997

§Department of Computer Engineering, Seoul Women's University, KOREA.

†Department of Electrical and Computer Engineering

‡Department of Information and Computer Science

University of California, Irvine, CA 92717

Email: kurdahi@ece.uci.edu

Phone: 1 (714) 824-8104

Fax: 1 (714) 824-2321

[1] **This work was supported by a MICRO grant from the University of California and Compass Design Automation Inc., and by a Fellowship from the Korea Organization of Science and Engineering Foundation. A preliminary version of this work appeared in the Proceedings of ICCAD-94 under the title: "Comprehensive Lower Bound Estimation from Behavioral Specifications".**

Abstract

A Unified Lower Bound Estimation Technique for High-Level Synthesis

The importance of effective lower bound estimation (LBE) techniques is well-established in High-Level Synthesis (HLS), since it allows more efficient exploration of the design space while providing other HLS tools with the capability of predicting the effect of specific tools on the design space. Much of previous work has focused on LBE techniques that use very simple cost models which primarily focus on the functional unit resources. With the push towards sub-micron technologies, simple models that use functional unit resources alone are not accurate enough to allow effective design space exploration since the effects of storage and interconnect can indeed dominate the cost function. In this paper, we present an integrated approach aimed at predicting lower bounds on hardware resources needed to implement a behavioral description within a given amount of time. Our area cost model accounts for storage (register) and interconnect resources (buses) in addition to functional resources. Our timing model uses a finer granularity that permits the modeling of functional unit, register and interconnect delays. Our approach is integrated because we consider the dependencies between the different types of resources as well as the ordering in which the resources are allocated. We tested our technique for functional unit, storage, and interconnect requirements on several high-level synthesis benchmarks and observed near-optimal results. We believe that our comprehensive LBE approach can lead to better quality HLS solutions in less time and demonstrate this approach in our paper.

Contents

List of Figures

List of Tables

1 Introduction

Estimation plays a central role in guiding the design tasks to optimal or near optimal solutions. While accurate estimation is somewhat important at physical and logic design tasks, it is even more crucial when the design process is started at a higher level of abstraction. Decisions made at this level do have a pronounced impact on the final design. However, the impact of these decisions cannot be found until later in the design process. Therefore, in order for such high level design tasks (mainly High Level Synthesis (HLS)) to produce reliable results, such tasks must rely on realistic and accurate models of hardware components. Without such realistic models, it is likely to produce designs not satisfying cost and/or timing constraints, resulting in unnecessary iterations through the design cycle and increasing the design turn around time.

Much of the earlier design prediction work assumed the existence of *netlist-based* design descriptions as inputs, and hence produced netlist-based estimators [1]. However, these techniques can only be used *after* the design data path is synthesized to provide *back-end* feedback. If, on the other hand, the designer starts with no feedback at all, or with incorrect feedback, then there is no guarantee that the design decisions initially made would indeed be the correct ones which would produce the desired outcome. Thus it is very important to provide the designer with *front-end* feedback to provide initial guidance in making design decisions. Specifically, we need to have the capability of *bounding* the design space prior to starting the HLS tasks.

In order to achieve this goal, we propose an integrated approach to predict *lower bound* estimates on resources, given a data flow graph (DFG) description of the design and a performance goal. Our approach is more comprehensive than the previous work in sense that our cost model accounts for registers and buses as well as FUs, and that we use an expanded timing model which can accommodate FU delays, register delay and interconnect delays altogether.

Whereas much of the previous work in the area has focused on techniques that use very simple cost models which primarily focus on the functional unit (FU) resources, our cost model accounts for storage (register) and interconnect resources (buses) in addition to functional resources. Our studies of state-of-the-art libraries of RT level components indicate that register cost can easily surpass those of "large" functional units such as adders. Table 1 shows such an example from the VTI 0.8μ cell library. According to this table, area of a register may indeed be larger than that of adder, implying that without including register cost, we cannot accurately predict the actual area cost. In addition to register cost, studies [2] have indicated that interconnect costs can be significant. Indeed, the move towards sub-micro technologies will result in interconnect costs being the dominant contributor towards design performance. Thus, we need to incorporate interconnects as well as registers in estimation for better prediction.

In addition, our timing model uses a finer granularity that permits the modeling of FU, register, and interconnect delays. Table 1 also indicates that the register delay cannot be ignored, so we need to consider register delay also. Our approach accommodates all of these factors in estimation along with transfer delays among these hardware resources such as bus-to-register delay, register-to-bus delay, FU-to-bus delay, bus-to-FU delay, FU-to-FU (chaining) delay, control delay, and so on. In our model, these delays are expressed in real time such as nanosecond (ns), and used along with the total performance constraint expressed in real time. It provides a more accurate analysis of the starting and completion times of each operation in the input data flow graph.

More importantly, our approach is integrated because we consider the dependencies between the different types of resources as well as the ordering in which the resources are allocated. This results in added flexibility since designers can constrain one type of resource and predict the remaining resources. Furthermore, our technique can predict the outcome of different HLS strategies having different allocation priorities. This allows the designer to further explore the design space, analyze resource allocation tradeoffs, and perform a true "what-if" analysis before committing to a particular strategy and a set of resources.

We have developed efficient algorithms and heuristics to support this model. Our initial experiments on some HLS benchmarks [3] indicate that this model is quite accurate. This estimation scheme naturally lends itself to encapsulation within system level synthesis frameworks by providing early and accurate estimates of design quality when large behavioral descriptions are partitioned onto several chips, without the need of running HLS tools to obtain full design netlists.

The rest of this paper is organized as follows. Section 2 describes previous and related work. Section 3 describes our architectural models and defines the problem at hand, and Section 4 gives an overview of the estimation algorithm. Sections 5, 6, and 7 describe the lower bound estimation techniques for functional unit, storage (register), and interconnect (bus) costs, respectively. Section 8 explains how to integrate these separate lower bound estimation techniques and to explore the design space for better estimates on the total area cost of hardware resources. Section 9 presents experimental results on several HLS benchmarks. Section 10 concludes with a summary.

2 Previous Work

Some of the recent work on estimating lower bounds on area cost and total control steps (or csteps) is described in [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. All of these works (with the exception of [5], [12], [13], and [14]) are mainly concerned with FUs in their area cost models. The work in [4] proposes a mathematical model for predicting the area-delay curve. An extension of this work is described in [5] and addresses lower bounds on time and area cost including interconnect cost, but not register cost. [6] proposes an ILP formulation for lower bound estimation of performance given resource constraints. [7] addresses lower bounds on time and FU cost for a functional pipelined data flow graph, but neither register cost nor interconnect cost. [9] also addresses lower bounds on time and FU cost. It uses these two estimation algorithms to predict the system level area-delay curve. However, it does not consider register and interconnect cost in estimation. [10] presents a formal approach which seems to estimate FU cost better than [5] and [6] in some benchmarks. It considers the interdependency of the bounds of different FU types, but neither registers nor interconnects in estimation. [11] finds the lower bound on FU cost and utilizes it in finding an optimal scheduling result effectively, but registers and interconnects are not considered. [12] uses an ILP formulation in calculating lower bounds on the number of FUs, registers, and buses separately. However, it does not take into account the dependencies among the number of resources of each type in estimating the lower bounds, and furthermore the solution can be computationally expensive. [13] presents an integrated area-delay prediction model which includes FU, register, and interconnect costs for use in system level partitioning. However, it does not consider interdependencies between hardware resources. Finally, [14] considers a generalized memory hierarchy scheme for a hardware/software co-design model and predicts the sizes of the various memory components to achieve a given performance goal.

3 Architectural Model and Problem Definition

In high-level synthesis, an RT-level system that consists of functional units, storages, and interconnects is synthesized from the behavioral description. We assume that each RTL operation can be executed using one type of RTL unit (e.g. an *add* operation can be executed by one type of adders).[2] In order to estimate accurately the number of resources required to implement the given behavior, we need to define a target architecture. In our approach, we consider two styles of target architectures as shown in Figure 1: multiplexor-based and bus-based architecture. Bus-based architectures are usually better suited for architectures with memories and register files, and are preferred due to the fact that interconnect is shared. In this paper, we confine our scope to bus-based architectural models (Figure 1(b)). On the other hand, estimating lower bounds on multiplexor-based architectures is much more difficult to handle due to the fact that one must consider the impact of several hard optimization steps aimed at reducing the number and size of multiplexors (e.g. switching the inputs of commutative operations). Such architectures will be considered in future work. We also assume that registers are used for storage resources. Registers can also be grouped into register files. In the bus-based architecture, we assume that each data transfer between an FU and a register occurs only through buses and that each data value produced by a functional unit should be stored in a register through buses so that it is used in a later clock cycle. However, the data value need not be stored in a register if its source and destination operations are chained.

[2] If more than one type of RTL units are available to execute a particular RTL operation, then we select the lowest cost one since this will lead to a lower bound on total area cost.

Since actual interconnect delays as well as its total area cost are dependent on the layout synthesis techniques such as placement and routing, we cannot easily estimate these figures. Therefore, it is currently assumed that users provide some initial estimates of these interconnect delays. Future linkage to physical level estimation tools will allow these delays to be more accurately estimated. Such linkage will also allow us to incorporate interconnects more accurately in the total area cost.

In our approach, the behavioral description, expressed in the form of a data flow graph, is given as input, and the total performance and clock period expressed in real time are given as constraints. In addition, FU delays, register delay, and interconnects delay are given by users or selected from a library. Given this information, we estimate lower bounds on the number of FUs of each type, on the number of registers, and on the number of buses (bi-directional or uni-directional).

4 Overview of Our Approach

4.1 Basic idea

Our estimation methods are based on the following principle: If N objects are distributed over K slots, at least $\lceil N/K \rceil$ objects are assigned to some slot within those K slots. This can be stated slightly differently if we are talking about a time slot interval $Z = [X, Y]$ whose length is $Y - X + 1$. If N_z objects are guaranteed to be assigned within interval Z, at least $\lceil N_z/(Y - X + 1) \rceil$ components are required to perform these objects. That is, this value can be used as the lower bound on the number of components to perform these objects. In our context, an object may represent an operation performed by an FU, a data value to be stored in a register, or a data transfer between hardware resources, while each time slot represent a control step (or clock cycle). To estimate tighter lower bounds, we compute these values over all the possible intervals, and then choose the maximum one as the lower bound.

While this principle has been used for FU estimation in prior work, in our work we have extended this approach for register and bus estimations as well.

4.2 The Area Cost Estimation Algorithm

In this paper, $ASAP_i$ $(ALAP_i)$ denotes the *earliest* (*latest*) cstep in which operation O_i can be *started* without violating both timing constraint and precedence relations between operations, and $ASAP'_i$ $(ALAP'_i)$ denotes the last cstep where operation O_i is *completed* when it is scheduled in $ASAP_i$ $(ALAP_i)$ cstep. In determining these values for each operation, we take into account the pre-defined transfer delay including the delays of registers and interconnects along with the delay of the FU itself, thus providing a more sophisticated timing model. In this paper, the cstep interval $[ASAP_i, ALAP_i]$ is called the *time frame* of operation O_i. We estimate the FU cost, register cost, and bus cost using these time frames. Our scheme, however, is flexible to allow modification of the estimation order among FUs, registers, and buses, allowing better design space exploration. We assume that each estimation step is independent for the purpose of explanation only. The integration of these estimation steps and the effects of the ordering will be described later in Section 8. Figure 2 shows the overall structure of the area cost estimation algorithm **LBE**, when FU cost is first estimated, register cost next, and bus cost finally.

We illustrate our approach with a walk-through example. Figure 3 shows an input DFG of the *differential equation* example which solves the 2nd order differential equation $y'' + 5xy' + 3y = 0$, and Figure 4 shows the initial time frames of the operations when the total performance (maximum delay) is 80 ns and the clock period is 20 ns. In this example, we assume that the total interconnect delay between an FU and a register is given as 4.5 ns, and that additions, comparisons, and subtractions are executed by ALUs with 10 ns delay and multiplications by multipliers with 15 ns delay.

5 FU Cost Estimation

5.1 Basic technique

If N operations of type T are scheduled in the interval Z, then *at least* $\lceil N/|Z| \rceil$ FUs of type T are required. Now given a particular operation O_i, then clearly, O_i is guaranteed to be scheduled in Z if its time frame $[ASAP_i,\ ALAP_i] \subseteq Z$. For each interval Z, we find the number of operations of type T guaranteed to be scheduled during that interval, and estimate the lower bound on the number of FUs of type T. After that, we enumerate all the cstep intervals $Z \subseteq [1, total\ cstep\ number]$ to get a tighter lower bound. The maximum such number over all enumerated intervals yields an estimated lower bound on the FU cost of type T.

Figure 5 illustrates how the lower bound on the number of FUs of each type is estimated. For example, the four multiplications O_1, O_2, O_3, and O_6 in Figure 4 are guaranteed to be executed in cstep interval $[1, 2]$, since their time frames are fully included in this interval. So $\lceil 4/2 \rceil = 2$ is estimated as a candidate lower bound on the number of multipliers for this interval. To get a tighter lower bound, these candidates are estimated over all the cstep intervals and the maximum one is selected. In this case, $max(0, 1, 2) = 2$ is finally chosen as the lower bound on the number of multipliers. In a similar way, the lower bound on ALU count is estimated as 2.

5.2 Generalization

The above basic idea is generalized to support multi-cycling, chaining, and functional pipelining of operations. Figure 6 shows an extended version of the FU cost estimation algorithm. In this algorithm,

$\psi(T)$ denotes a set of operations of type T, and $P_{i,z}$ ($Q_{i,z}$) is the intersection of interval Z and the *earliest* (*latest*) cstep interval during which O_i may be performed by an FU of type T. O_i is guaranteed to occupy an FU for as many csteps as $L_{i,z}$ *at least* in the interval Z. Therefore, $\sum_{O_i \in \psi(T)} L_{i,z}$ represents the total number of FU slots required in the interval Z, and $\lceil (\sum_{O_i \in \psi(T)} L_{i,z})/|Z| \rceil$ yields a lower bound on the number of FUs of type T estimated for interval Z. We enumerate all the cstep intervals Z to get a tighter lower bound, and then select the maximum one as the lower bound on the number of FUs of type T.

For each non-pipelined operation O_i with delay of less than or equal to one cstep, $ASAP'_i$ and $ALAP'_i$ are exactly the same as $ASAP_i$ and $ALAP_i$ respectively. Therefore, we can apply the same algorithm for chained operations as well as multi-cycling operations without any modification.

As for the pipelined operations, we may have to count the number of slots required for each stage in order to estimate the number of pipelined FUs, assuming that each *k*-stage pipelined FU is divided into *k* separate operations. In this algorithm, however, we only have to consider the first stage of them in estimating the lower bound on the number of the pipelined FUs. This is because the number of slots required for stage *j* in interval $Z_1 = [X, Y]$ is less than or equal to the number of slots required for the first stage in another interval $Z_2 = [max(1, X - j + 1), Y - j + 1]$ while $|Z_2| \leq |Z_1|$. Therefore, for each pipelined operation O_i, we assume that $ASAP'_i$ and $ALAP'_i$ are equal to $ASAP_i$ (i.e. $ASAP'_i$ of the first stage) and $ALAP_i$ (i.e. $ALAP'_i$ of the first stage) respectively, and then apply the same algorithm.

An estimated lower bound on the total FU cost is derived by applying the above procedure for all the types.

5.3 Refinement of the FU estimates

Sometimes our estimated lower bounds are loose since our search may include some non-feasible situations. However, we can use those bounds as an initial solution which is further refined to obtain tighter lower bounds. Figure 7 shows how to tighten the lower bound. We use the lower bound obtained so far as the initial solution and iteratively apply a relaxation based technique to improve the solution into a tighter Lower Bound.

Basically, we assume that each FU is used as many as its initial lower bound, and then adjust the time frames of operations using this FU constraint. We temporarily schedule each operation O_i into its ASAP cstep, and then calculate the lower bound again. When estimating the lower bound again, we may need to consider all the possible intervals in $[1, total\ cstep\ number]$ to get a tighter bound. In this procedure, however, we consider only one cstep interval instead of considering all the possible intervals to improve the speed of the re-estimation. We select interval $[1, ASAP_i]$ in this procedure, since only the time frames of the predecessors of O_i may be affected by scheduling O_i into $ASAP_i$. LB'_T represents the re-estimated lower bound on the number of FUs of type T for the interval $[1, ASAP_i]$. Once the new lower bound LB'_T is estimated, we compare it with the FU constraint. If the new one is larger than that constraint, it implies that O_i could not be scheduled into $ASAP_i$ under the FU constraint, and thus we need not include $ASAP_i$ in O_i's time frame. Therefore, in that case, we can increment $ASAP_i$ and $ASAP'_i$ by one. In a similar way, we adjust the $ALAP$ cstep of each operation.

For example, the initial time frame of O_3 is $[1, 2]$ as shown in Figure 4. However, if we assume that the number of multipliers available is equal to the lower bound on the number of multipliers($= 2$),

O_3 cannot be scheduled into cstep 1, since at least 2 other multiplications O_1 and O_2 are guaranteed to be scheduled into cstep 1. Therefore, the time frame of O_3 shrinks to [2,2]. In a similar way, we can adjust the $ALAP$ times of the operations. Figure 8 shows the time frames of the operations modified by this update procedure. In this example, time frames of operations O_3, O_4, O_5, O_7, O_8, and O_9 are modified compared with those in Figure 4.

Once the time frames are adjusted, we try to see if both the FU constraint and the performance constraint already set are satisfied by checking whether all the time frames are still *valid*[3] or not. If not, then we need to relax the FU constraint. So we increment the lower bound by one and repeat above procedure. In other cases, we estimate a new lower bound again using the adjusted time frames. If we can find a tighter lower bound, it contradicts the assumption that as many FUs are used as the initial lower bound. So, we increment the initial lower bound by one and repeat above procedure with this increased lower bound as the new FU constraint.

5.4 Time complexity of the FU cost estimation algorithm

The time complexity in estimating the initial FU cost is $O(N \cdot C^2)$ and that in refining the estimation is roughly $O(N \cdot (C^2 + N + E))$, where N and E represent the number of operations and the number of edges in the given DFG respectively, and C the total number of csteps. Thus the total time complexity of this algorithm is $O(N \cdot (C^2 + N + E))$.

[3] We say that the time frame of operation O_i is *valid* only if $ASAP_i \leq ALAP_i$.

6 Register Cost Estimation

6.1 Basic technique

The main difficulty in estimating the register cost arises from the fact that *no prior scheduling is assumed*. This means that the lifetime of each data value (variable) is not known *a priori*, since it is only known once scheduling is performed. Therefore, the estimation approach should consider all the possible lifetimes of all the variables.

Figure 9 shows our register cost estimation algorithm. In this algorithm, $V_{i,j}$ denotes a data value from operation O_i to O_j. The operation O_i is called the *source* of variable $V_{i,j}$, and the operation O_j the *destination* of the variable. The *weight* of variable $V_{i,j}$, denoted by $W_{i,j}$, represents the *minimum* size of lifetimes of the variable. It is obtained by analyzing the critical path between the source and the destination of the variable. That is, once variable $V_{i,j}$ becomes *active*, it should remain active for *at least* $W_{i,j}$ contiguous csteps. If the weight of each variable is determined, we compute $\sum_{V_{i,j}} M_{i,j,z}$ for each interval Z, where $M_{i,j,z}$ is the portion of variable $V_{i,j}$'s lifetime guaranteed to be included in the interval Z, taken over all possible schedules. Since $\sum_{V_{i,j}} M_{i,j,z}$ is the total number of register slots required for interval Z, $\lceil\ (\sum_{V_{i,j}} M_{i,j,z})\ /\ |Z|\ \rceil$ represents the minimum number of registers required for that interval, that is, a lower bound on the register count. We enumerate all the cstep intervals $Z \subseteq [1, total\ cstep\ number + 1]$ to get a tighter lower bound and then select the maximum one as the lower bound on the register count.

6.2 Improvements

This basic technique, however, suffers from a serious drawback: in most cases, the weight of a variable is usually too *small*, and thus this basic procedure would yield a trivial lower bound on the register cost.

In order to alleviate this problem, we need to find the largest possible weights for the variables. In this paper, we apply two improvement techniques called *Fanout Reduction* and *Variable Merging*. Table 2 shows the variables and their weights before and after these improvement techniques are applied for the differential equation example.

6.2.1 Fanout reduction

Consider two shared variables $V_{i,j}$ and $V_{i,k}$, which have the same source O_i but different destinations. If one destination O_j is guaranteed to be completed before another destination O_k, then $V_{i,j}$ does not need to be considered when estimating the register count, since the lifetime of $V_{i,j}$ is always included in that of $V_{i,k}$. For example, if $ALAP'_j \leq ASAP'_k$ or O_k is data dependent on O_j, then we only have to consider $V_{i,k}$ in our basic procedure. This technique will help reduce the total problem size and also simplify the variable merging problem explained later.

Figure 10 illustrates how the fanout reduction technique is applied to a simple example. As an example, the two fanouts a and a' in this figure have the same source O_1, but different destinations: O_3 and O_4 respectively. Since "$ALAP'_4 \leq ASAP'_3$" is satisfied regardless of scheduling results when the total number of csteps is 4, the lifetime of fanout a' always includes that of the other fanout a. So we do not consider the fanout a' for estimation purpose only. In a similar way, we do not consider fanout b' which is covered by fanout b.

6.2.2 Variable merging

Since our goal is only in counting *the number of registers*, we do not need to assume a particular register binding. In this case, the lifetimes of some variables can be merged for the purpose of estimation only.

For example, if O_j is not pipelined, two variables $V_{i,j}$ and $V_{j,k}$ can be merged as a new variable, say $V_{i,k}$, with a larger weight, since they cannot be active at the same time and their merged lifetime covers their previous lifetimes. This modification helps increase $\sum_{V_{i,j}} M_{i,j,z}$ in Figure 9 and also reduces the number of variables or problem size. In this technique, shared variables are not considered for merging. However, by the fanout reduction procedure described above, we can reduce the number of shared variables.

An example of variable merging is shown in Figure 10. Note that neither a nor c are *individually* guaranteed to be included in interval $[3, 3]$, since operation O_3 can be scheduled in cstep 2 or 3. However, we can easily recognize that *one* of these two variables (either a or c) is guaranteed to be active in interval $[3, 3]$ regardless of scheduling results of operation O_3. This implies that at least one register is required for one of these two data values (either a or c) during interval $[3, 3]$. Therefore, we merge the two variables a and c are merged into a new variable, say ac. Note that this merging is done in order to improve the estimation quality and does not necessarily imply an assignment of values to registers. In a similar way, variables 'd and e' or 'b and e' can be merged.

6.3 Refinement of the register estimate

As in the FU estimation case, the initial lower bound on register count can be refined further to obtain a tighter lower bound. In this case, the maximum number of values which can be active during the same cstep is restricted to the initial lower bound on register count. Figure 11 shows the algorithm which refines the register estimate. Note that this algorithm is very similar to that in Figure 7 which refines the lower bounds on FU counts. In this algorithm, however, registers are assumed to be used as many times as the initial lower bound on register count.

6.4 Register Files

If registers are grouped into a register file, then the estimation can still be used to estimate the register file size. However, there may be some restrictions on the maximum number of simultaneous accesses to the register file. Say the available register files can have at most n ports. For a given performance constraint, our algorithms can estimate the maximum number of simultaneous accesses to memory over all control steps, *m*. Then we can estimate that at least $Est\_RegisterFiles = \lceil m/n \rceil$ register files are needed to accommodate the performance requirements. Since no assignment of values to storage has been accomplished, we must assume that storage is distributed equally among the register files. Thus, each register file will have a size of $Est\_RegisterFile\_Size = \lceil Est_Register_Cost/Est\_RegisterFiles \rceil$.

6.5 Time complexity of the register cost estimation algorithm

The time complexity is $O(E \cdot (N + E))$ in calculating the weights of variables and in the improvement steps, $O(E \cdot C^2)$ in estimating the initial register cost, and $O(E \cdot (C^2 + N + E))$ in refining the register cost, where N represents the number of operations, E the number of values to be stored in registers, and C the total number of csteps. Thus the total time complexity of this algorithm is $O(E \cdot (C^2 + N + E))$.

7 Bus Estimation

7.1 Basic technique

In this paper, we assume that each data transfer between an FU and a register occurs only through buses. Therefore, the number of buses required is determined by the maximum number of concurrent data transfers via buses. We find the maximum number of concurrent data transfers from registers to FUs and then that from FUs to registers.

Figure 12 shows our bus cost estimation algorithm. In this algorithm, Γ is defined as the set of data values which should be stored in registers. A variable whose source and destination can be chained into the same cstep need not be stored in a register; such a variable is not included in Γ. $\Psi_I(Z)$ denotes the set of data values to be transferred from registers to FUs during cstep interval Z, and $\Psi_O(Z)$ represents the set of data values generated by FUs and transferred to be stored in registers during interval Z. Therefore, $\lceil\ |\Psi_I(Z)|\ /\ |Z|\ \rceil$ represents a lower bound on the number of data transfers from registers to FUs during interval Z. Similarly, $\lceil\ |\Psi_O(Z)|\ /\ |Z|\ \rceil$ represents a lower bound on the number of data transfers from FUs to registers during interval Z. We enumerate all the cstep intervals $Z \subseteq [1, total\ cstep\ number]$ to get a tighter lower bound and then select the maximum value over all such intervals as the lower bound on the corresponding bus count.

$\Psi_I(Z)$ in the algorithm, however, may include more than two different fanout variables which represent the same data value. If such data values are used during the same cstep, we may need only one data transfer for those values, reducing the size of concurrent bus access. In order to detect such the fanout values, we analyze the time frames of the destinations of the fanout variables, and determine whether they always use the data value at the same cstep.

The total number of buses can be estimated using these two lower bounds according to the bus style in the target architecture: separate uni-directional buses (Input buses and Output buses) or bi-directional buses (I/O buses). We add the two lower bounds for the uni-directional bus style, while choosing the maximum of them for the bi-directional bus style.

7.2 Refinement of the bus estimates

Just like the refinement of other resource estimation, the initial lower bounds on bus counts can be refined further to obtain tighter lower bounds. In this procedure, however, as many input/output buses are assumed to be used as the initial lower bound. Figure 13 shows the algorithm for tightening the bus estimates. This algorithm is very similar to that in Figure 7 and that in Figure 11 which are used for refining the lower bounds on FU counts and on register count respectively. In this algorithm, however, buses are assumed to be used as many as the initial lower bound on bus count.

7.3 Time complexity of the bus cost estimation algorithm

The time complexity is $O(C^2 \cdot (E + N^2))$ in obtaining $\Psi_I(Z)$ and $\Psi_O(T)$ for all the intervals and $O(N \cdot (C^2 + N + E))$ in refining the bus cost, where N represents the number of operations, E the number of values to be stored in registers, and C the total number of csteps. Thus the total time complexity of this algorithm is $O(C^2 \cdot (E + N^2) + N \cdot E)$.

8 Integrated Resource Estimation

In the previous sections, we assumed that the estimation algorithms for each class of resources (i.e. FUs, registers, or buses) are applied independently given a data flow graph and timing constraints. However, since resource requirements are interdependent, such a strategy could lead to overall estimates representing non-feasible solutions. Thus, we need to consider the dependencies between the different types of resources to get more realistic estimates.

As an example, consider the simple data flow graph shown in Figure 14(a), which consists of 2 additions, 1 multiplication, and 5 values to be stored in registers. For this example, there exist only 2 possible schedules shown in Figure 14(b) and 14(c) respectively, if the total cstep number is 2. The schedule in Figure 14(b) requires 2 adders, 1 multiplier, and 2 registers, whereas that in Figure 14(c) requires 1 adder, 1 multiplier, and 3 registers. This means that the lower bounds on the numbers of adders and multipliers are 1 respectively and that on register count is 2, since we consider all the possible schedules in our estimation. However, there is no possible schedule which can be implemented by 1 adder, 1 multiplier, and 2 registers. Therefore, in this case, it is more realistic to estimate 1 adder, 1 multiplier, and 3 registers as lower bounds if the area cost of register unit is less than that of adder, or to estimate 2 adders, 1 multiplier, and 2 registers if the area cost of register unit is larger than that of adder.

In order to obtain more realistic lower bounds, we estimate a *conditional* lower bound on a type of resource subject to other types of resources rather than an *absolute* lower bound. We assume that resources of a particular type are allocated as many times as the estimated lower bound at the estimation step for that type. This constraint may restrict the time frames of the operations and the lifetimes of variables, and thus affect the estimation of other resources.

Consider the example in Figure 14 again, assuming that the FU cost estimation is followed by register cost estimation. When we estimate the register cost for that example, we assume that 1 adder and 1 multiplier are used, since their lower bounds estimated in the FU cost estimation step are 1 each. This constraint forces addition operation O_2 to be scheduled in cstep 2, and also fixes the lifetimes of input/output values of O_2. >From the fixed lifetimes, our algorithm estimates 3 registers as the lower bound on storage.

9 Experimental Results

9.1 Implementation

We implemented our integrated estimation approach in the C language on a SUN Sparcstation. The structure of the system is shown in Figure 15. The behavioral description of the target design is specified in the form of Data Flow Graph (DFG). A constraint on the total execution time is also specified by the user. A library of components provide the area and delay data for each component, including registers and estimates of bus delay.

Since our integrated approach incorporates the dependencies between the different types of resources, the ordering of estimation is very important. Such an ordering may be determined by the area cost of each resource type or by users. By default, the system assumes that the resource type with largest area cost is estimated first, then the second largest, and so on. Our scheme provides flexibility to modify the estimation order, so designers can constrain one type of resource and predict the remaining resources subject to that constraint. Furthermore, our technique can predict the outcome of different HLS strategies having different allocation priorities. Additionally, the designer may choose to individually constraint each type of resource, and run the system to estimate the remaining resources subject to his/her constraints. This allows the user to further explore the design space, analyze resource allocation tradeoffs, and to perform true "what-if" analysis prior to running the synthesis tasks. Once resource estimates are obtained, the user may use those as constraints to run the synthesis tools themselves, thus ensuring predictable results, and reducing the overall number of iterations in running the synthesis tools. This flexibility is unique to our estimation approach.

9.2 Experimental Setup

In order to validate our lower bound estimation system, we applied it to some well-known high level synthesis benchmarks. The first example is *the 2nd order differential equation solver* [3] which consists of 6 multiplication operations, 2 additions, 2 subtractions, and 1 comparison. In this example, we assumed that multiplications are performed by multipliers while additions, subtractions, and comparison are performed by ALUs. The second example is *the 5th order elliptic wave filter* (EWF) [3] which consists of 8 multiplications and 26 additions, and the third example is *the AR filter* [21] which consists of 16 multiplications and 12 additions. In these two examples, multiplications are assumed to be performed by multipliers and additions by adders. The fourth example is the *Discrete Cosine Transformer* (DCT) consisting of 16 multiplications, 25 additions, and 7 subtractions. The final example is the *Jacobian Transformer* [23]. It is a large example consisting of 396 multiplications, 252 additions, 18 subtractions, and 72 memory accesses for table look-up. In these two examples, we assumed that multiplications are performed by multipliers while additions and subtractions by ALUs.

Our delay and timing assumptions were as follows: we assume that the clock period is 20 ns, the maximum transfer delay between an FU and a register is 4.5 ns, and the chaining delay (transfer delay between two FUs when they are chained) is 0.2 ns. Some design information are specified in the tables. If not explicitly specified, the library in Table 1 is assumed to be used. We also assume that area cost and delay of a pipelined multiplier are the same as those of a non-pipelined, respectively. The CPU time for each experiment ranges from 0.1 sec to 1 min. on a SUN 4/600 workstation.

9.3 Experiment 1 – Effects of refinement steps

A first set of experiments was performed to determine the effect of refinement steps. Table 3 shows the effects of our refinement steps. In this table, estimation results with and without refinement steps are shown respectively. As an example, when the performance constraint is set to 120 ns, 1 ALU and 3 multipliers are estimated without our refinement step for the differential equation circuit, but one more ALU is estimated WITH our FU refinement step. Moreover, one more register is estimated with the register refinement step. Notation x/yb denotes the lower bound on number of input buses is x while that of output buses is y.

9.4 Experiment 2 – Effects of ordering on estimation results

A second set of experiments was performed to gauge the effect of changing the order of estimation on different examples. Since there are three classes of resources (FU, Register, and Bus), there can be six orderings of the estimation sequence. For each benchmark, we first set up an initial constraint on execution time. Next, we ran the system six times each with a different estimation order. This is repeated for different constraints on execution time. Tables 4 through 8 summarize our estimation results on these benchmarks and the effects of changing the ordering of estimation. In these tables, a character string consisting of 'F', 'R', and 'B' implies the ordering of the estimations. For example, "FRB" means FUs are estimated first, then Registers with FU constraints, and finally Buses with FU and register constraints. In these experiments, we assumed that all input and output data values are considered to be stored in registers.

As shown in these tables, the ordering of the estimations does affect the estimation results in many cases. Table 4, for example, shows that 2 adders, 3 multipliers, 6 registers, and 4 bi-directional buses are estimated if FU resources are estimated first, while 1 adder, 4 multipliers, 7 registers, and 3 buses are estimated if buses are estimated first, when the maximum delay is 120 ns. This result implies that it takes a higher bus cost to implement the given behavior with less FU cost or conversely it may take a higher FU/register cost to minimize the bus cost. In contrast, in Table 5, we obtained the same results regardless of the estimation order in most cases (except when the total performance is 420 ns), implying that this example may have less potential trade-offs among FUs, registers, and buses. Using our approach, users can explore the design space by attempting different orderings and observing the results.

9.5 Experiment 3 – Assessing the quality of the estimates

To demonstrate the quality of our estimation, we compared our results with actual designs obtained through conventional scheduling and allocation processes. We compared our results with actual designs generated by HAL [17], InSyn [18], ALPS [19], OASIC [12], ILP [20], and SALSA [22]. Tables 9 through 14 show the comparison results. Note that some of these systems did not optimize any resources other than Functional Units, in many cases resulting in designs which may be sub-optimal with respect to registers. In such cases, we attempted to improve the FU-optimal designs by generating some so-called "Hand-Optimized Designs". These "Hand-Optimized Designs" are obtained by generating several schedules that are optimal in the number of FUs using our scheduling system [11], and then choosing the one with the minimum number of registers. We generated such designs only when we concluded that designs reported in literature optimized only FUs or we could not find any comparable designs published in the literature.

Since previous synthesis systems used a fixed ordering of allocation (i.e., FUs first, then registers, and buses finally), we applied our estimation in the same order to enable comparison. We note, however, that some synthesis systems do not report results on all three classes of resources. For example, ALPS has no experimental results on registers, while ILP has no results on buses. However, they are described here just for comparison with our estimation results on the remaining resources.

Overall, these experimental results clearly show that our estimates are quite accurate. Indeed, the FU estimates are exact in most cases. Furthermore, the estimated lower bounds on register and bus count are also quite accurate as shown in Tables 9 through 14. In most cases, the lower bound on register (bus) cost is 1 or 2 units below the actual register (bus) count.

As an example, in Tables 10 and 11, the numbers of FUs of each type and registers in our estimation are the same as those of actual designs generated by OASIC and InSyn. It implies that, for this example, OASIC and InSyn generated designs which are optimal in terms of FUs and registers respectively and that our estimation is very accurate as well, especially given that no prior scheduling is assumed in our estimation. On the other hand, other systems such as SALSA, do not optimize the number of register, and hence the reported designs have more registers that what it predicted by the estimation.

In order to investigate the importance of considering registers as well as FUs in estimating design cost, we show in Figures 16, 17, 18, 19, 20, and 21 some of the comparison results in a graphical form. In these figures, X-axis shows the total performance or maximum delay and Y-axis compares the area cost occupied by FUs and registers for the designs in Tables 9 through 14 and the corresponding estimation results. [4]. In addition, "FU-only Models" show the area estimates obtained by accounting

[4] We did not include bus cost in the total area cost since it is not easy to quantify the area of buses before physical level synthesis.

for FUs only. These charts clearly show that by using the FU-only model, estimations are highly inaccurate, and thus verify the importance of register estimation.

10 Conclusions

In this paper, we motivated the need for a more comprehensive lower-bound estimation algorithm that takes into account not only functional unit costs, but also the costs of storage and interconnects. We also motivated the need to account for a better characterization of time (in terms of real nanoseconds per clock), as opposed to traditional methods that only considered the number of clock cycles. Our estimates of hardware resource requirements are quite accurate and validate our approach for these examples. Our approach is integrated and flexible, and accounts for the dependencies in the ordering of the FU, register, and bus estimation. Thus, our system allows easy exploration of the design space, and evaluation of alternatives prior to committing to an architecture.

Currently, our system does not consider multiplexor and wiring area and delay in its estimation. In addition, it does not allow loops and branches in the input behavioral description. These topics and others will be addressed in future work.

References

[1] C. Ramachandran, F. J. Kurdahi, D. Gajski, V. Chaiyakul, and A. Wu, “Accurate Layout Area and Delay Modeling for System Level Design,” *Proc. ICCAD ’92*, Nov. 1992.

[2] M. C. McFarland, “Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions,” *Proc. 23rd DAC*, pp. 474-480, July 1986.

[3] N. Dutt and C. Ramachandran, “Benchmarks for the 1992 High Level Synthesis Workshop,” *Tech. Report #92-10*7, Information & Computer Science Department, UC Irvine, 1992.

[4] R. Jain, A. C. Parker, and N. Park, “Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs,” *IEEE Trans. CAD*, vol 11. no. 8, pp. 955-965, August 1992.

[5] A. Sharma and R. Jain, “Estimating Architectural Resources and Performance for High-Level Synthesis Applications,” *IEEE Trans. VLSI Systems*, vol 1. no. 2, pp. 175-190, June 1993.

[6] M. Rim and R. Jain, “Estimating Lower-Bound Performance of Schedules Using a Relaxation Technique,” *Proc. ICCD ’92*, pp. 290-294, Oct. 1992.

[7] Y. Hu, A. Ghouse, and B. S. Carlson, “Lower Bounds on the Iteration Time and the Number of Resources for Functional Pipelined Data Flow Graphs,” *Proc. ICCD ’93*, pp. 21-24, 1993.

[8] Y. Hu and B. S. Carlson, “Improved Lower Bounds for the Scheduling Optimization Problem,” *Proc. ISCAS ’94*, pp. 295-298, 1994.

[9] A. H. Timmer, M. J. M. Heijligers, and J. A. G. Jess, “Fast System-Level Area-Delay Curve Prediction,” *Proc. 1st APCHDL*, pp. 198-207, 1993.

[10] Samit Chaudhuri and Robert A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," *Proc. 7th International Symposium on High-Level Synthesis*, pp. 36-41, May 1994.

[11] Seong Y. Ohm and Chu S. Jhon, "A Branch and Bound Method for the Optimal Scheduling," *Proc. CICC '92*, May 1992.

[12] C. H. Gebotys and M. I. Elmasry, "Simultaneous Scheduling and Allocation for Cost Constrained Optimal Architectural Synthesis," *Proc. 28th DAC*, pp. 2-7, June 1991.

[13] Kayhan Küçükçakar, "System-Level Synthesis Techniques with Emphasis on Partitioning and Design planning," *PhD Thesis*, EE-systems Dept., USC, Sept. 1991.

[14] P. Gupta and A. C. Parker, "SMASH: A Program for Scheduling Memory-Intensive Application-Specific Hardware," *Proc. 7th International Workshop on HLS*, pp. 54-59, May 1994.

[15] Seong Y. Ohm, "A Study on the Scheduling Method based upon Lower Bound Estimation", *Ph.D. Dissertation*, Department of Computer Engineering, Seoul National University, Aug. 1992.

[16] A. H. Timmer and J. A. G. Jess, "Execution Interval Analysis under Resource Constraints," *Proc. ICCAD '93*, pp. 454-459, Nov. 1993.

[17] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. CAD*, vol. 8, no. 6, pp. 661-679, June 1989.

[18] A. Sharma and R. Jain, "InSyn: Integrated Scheduling for DSP Applications," *Proc. 30th DAC*, pp. 349-354, June 1993.

[19] J. H. Lee, Y. C. Hsu and Y. L. Lin, “A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis,” *Proc. ICCAD-89*, pp. 20-23, November 1989.

[20] S. Chaudhuri, R. Walker, and J. Mitchell, “Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem,” *Tech. Report #94-15*, Department of Computer Science, Rensselaer Polytechnic Institute, July 1994.

[21] S. Y. Kung, “On Supercomputing with Systolic/Wavefront Array Processors,” *Proc. IEEE*, vol. 72, no. 7, pp. 867-884, July 1984.

[22] J. A. Nestor and G. Krishnamoorthy, “SALSA: A New Approach to Scheduling with Timing Constraints,” *IEEE Trans. CAD*, vol. 12, no. 8, pp. 1107-1122, August 1993.

[23] S. Bakshi and D. Gajski, “A strategy for design space exploration,” *Tech. Report #93-10*, Information & Computer Science Department, UC Irvine, 1993.

Unit	Area (μ^2)	Delay (ns)	Library Name
Adder	40,000	15.0	DPADD001H (16 bit)
Multiplier	58,000	24.4	DPMLT011H (16 bit)
Register	41,440	2.17	DPDFF0801 (16 bit)

Table 1: Area and delay figures for adders, multipliers, and registers from the VTI 0.8μ data path library.

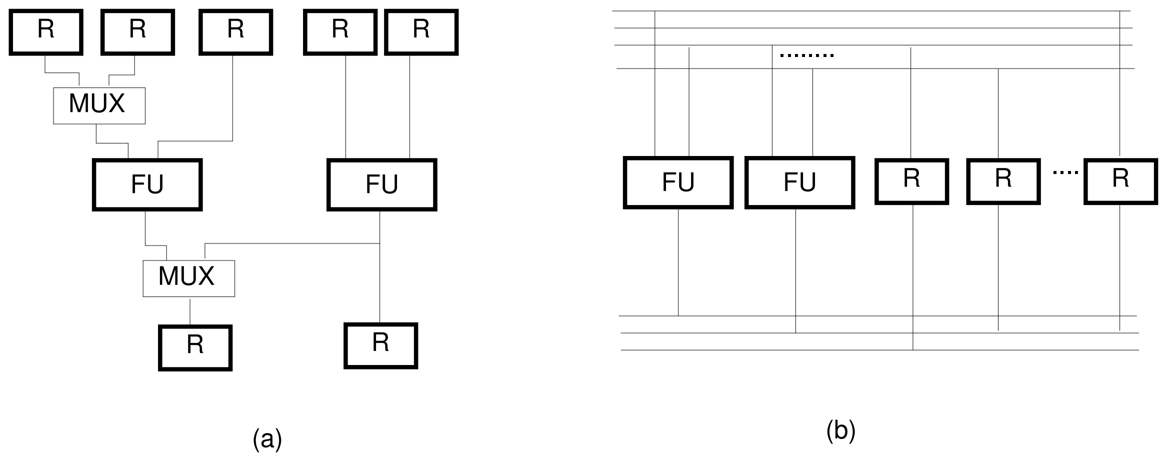

Figure 1: Architectural models: (a) multiplexor-based architecture and (b) bus-based architecture.

```
LBE()
{
        parse input DFG;
        read delay and area information;
        read clock period and total performance;
        total cstep number = ⌊ (total performance) / (clock period) ⌋;

        for each operation O_i in input DFG,
                determine ASAP_i, ALAP_i, ASAP'_i, ALAP'_i;

        Est_FU_Cost = Estimate_FU_Cost();
        Est_Reg_Cost = Estimate_Reg_Cost();
        Est_Bus_Cost = Estimate_Bus_Cost();

        Est_Area_Cost = Est_FU_Cost + Est_Reg_Cost + Est_Bus_Cost;

        return(Est_Area_Cost);
}
```

Figure 2: One formulation of our area cost estimation algorithm.

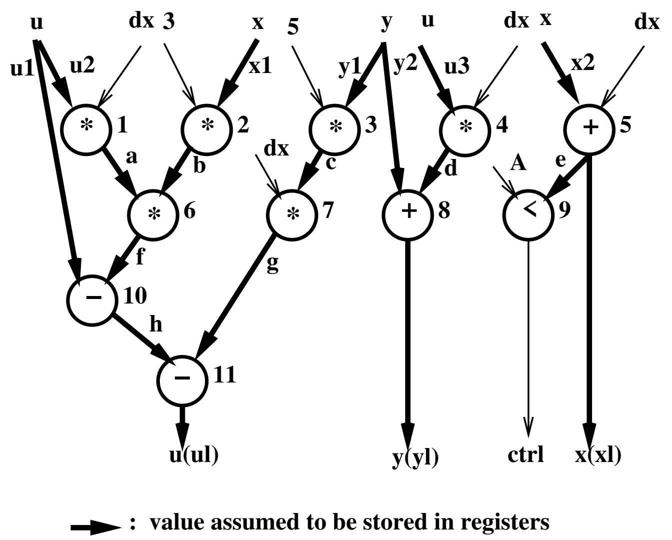

Figure 3: A DFG of the 2nd order differential equation example.

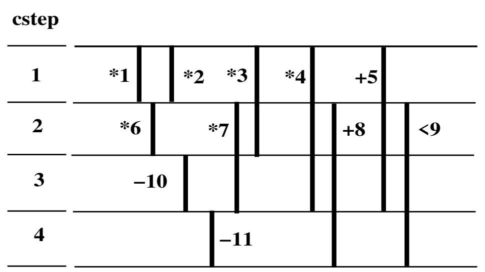

Figure 4: The initial time frames of the operations.

Multiplier $(*)$

$[1,1] : \{O_1, O_2\} \Rightarrow \lceil 2/1 \rceil = 2$
$[1,2] : \{O_1, O_2, O_3, O_6\} \Rightarrow \lceil 4/2 \rceil = 2$
$[1,3] : \{O_1, O_2, O_3, O_4, O_6, O_7\} \Rightarrow \lceil 6/3 \rceil = 2$
$[1,4] : \{O_1, O_2, O_3, O_4, O_6, O_7\} \Rightarrow \lceil 6/4 \rceil = 2$
$[2,2] : \{O_6\} \Rightarrow \lceil 1/1 \rceil = 1$
$[2,3] : \{O_6, O_7\} \Rightarrow \lceil 2/2 \rceil = 1$
$[2,4] : \{O_6, O_7\} \Rightarrow \lceil 2/3 \rceil = 1$
$[3,3] : \{\} \Rightarrow 0$
$[3,4] : \{\} \Rightarrow 0$
$[4,4] : \{\} \Rightarrow 0$

$LB_* = max(0, 1, 2) = 2$
$Est\_FU\_Cost = 2 \times Area_* + 2 \times Area_{ALU}$

ALU $(+, -, <)$

$\{\} \Rightarrow 0$
$\{\} \Rightarrow 0$
$\{O_5, O_{10}\} \Rightarrow \lceil 2/3 \rceil = 1$
$\{O_5, O_8, O_9, O_{10}, O_{11}\} \Rightarrow \lceil 5/4 \rceil = 2$
$\{\} \Rightarrow 0$
$\{O_{10}\} \Rightarrow \lceil 1/2 \rceil = 1$
$\{O_8, O_9, O_{10}, O_{11}\} \Rightarrow \lceil 4/3 \rceil = 2$
$\{O_{10}\} \Rightarrow \lceil 1/1 \rceil = 1$
$\{O_{10}, O_{11}\} \Rightarrow \lceil 2/2 \rceil = 1$
$\{O_{11}\} \Rightarrow \lceil 1/1 \rceil = 1$

$LB_{ALU} = max(0, 1, 2) = 2$

Figure 5: An example illustrating how the lower bounds are estimated.

```
Estimate_FU_Cost()
{
    for each FU type T,
        ψ(T) = set of operations of type T;
        for each cstep interval Z ⊆ [1, total cstep number],
            for each operation O_i in ψ(T),
                P_{i,z} = [ASAP_i, ASAP'_i] ∩ Z;
                Q_{i,z} = [ALAP_i, ALAP'_i] ∩ Z;
                L_{i,z} = min(|P_{i,z}|, |Q_{i,z}|);
        LB_T = max_z (⌈ (Σ_{O_i ∈ ψ(T)} L_{i,z}) / |Z| ⌉);
    Est_FU_Cost = Σ_T (LB_T × Area_T);
    update Est_FU_Cost;
    return(Est_FU_Cost);
}
```

Figure 6: Our FU cost estimation algorithm.

```
Update_LB_on_FU()
{
    For each type T,
  loop:
        FU_T = LB_T;
        Adjust_Time_Frames_with_FU_Constraint(FU_T);
        if there is an operation O_k in ψ(T) such that ASAP_k > ALAP_k,
            increment LB_T and go to loop;
        calculate new lower bound LB'_T using the temporary time frames;
        if LB'_T > FU_T, increment LB_T and go to loop;
}

Adjust_Time_Frames_with_FU_Constraint(FU_T)
{
    For each operation O_i in ψ(T),
        Z = [1, ASAP_i];
        Schedule O_i temporarily to ASAP_i and then temporarily
            adjust the ALAP csteps of its predecessors accordingly;
        LB'_T = ⌈ (Σ_{O_i ∈ ψ(T)} L_{i,z}) / |Z| ⌉;
        Resume all the time frames;
        if(LB'_T > FU_T), increment ASAP_i and ASAP'_i by one;

        Z = [ALAP_i, total cstep number];
        Schedule O_i temporarily to ALAP_i and then temporarily
            adjust the ASAP csteps of its successors accordingly;
        LB'_T = ⌈ (Σ_{O_i ∈ ψ(T)} L_{i,z}) / |Z| ⌉;
        Resume all the time frames;
        if(LB'_T > FU_T), decrement ALAP_i and ALAP'_i by one;
}
```

Figure 7: Algorithm for refining the FU estimates.

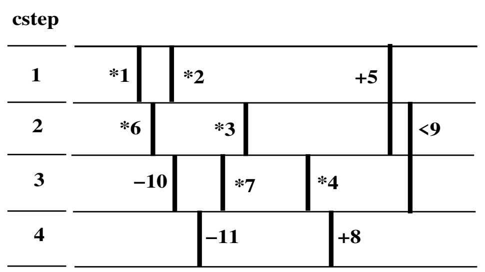

Figure 8: The adjusted time frames of the operations.

Estimate_Reg_Cost()

{

for each variable $V_{i,j}$, calculate $W_{i,j}$;
Fanout_Reduction();
Variable_Merging();
for each interval $Z \subseteq [1, total\ cstep\ number + 1]$,
    for each variable $V_{i,j}$,
        if destination O_j is *pipelined*, $S_j = ASAP_j$;
        else $S_j = ASAP'_j$;
        $P_{i,j,z} = [S_j - W_{i,j} + 1, S_j] \cap Z$;
        $Q_{i,j,z} = [ALAP'_i + 1, ALAP'_i + W_{i,j}] \cap Z$;
        $M_{i,j,z} = min(|P_{i,j,z}|, |Q_{i,j,z}|)$;
$LB_{Reg} = max_z\ (\lceil\ (\sum_{V_{i,j}} M_{i,j,z})\ /\ |Z|)\ \rceil)$;
$Est\_Reg\_Cost = LB_{Reg} \times Area_{Reg}$;
update Est_Reg_Cost;
return(Est_Reg_Cost);

}

Figure 9: Our register cost estimation algorithm.

Variables and Weights before Improvements									
variable	u_1	u_2	u_3	x_1	x_2	y_1	y_2	ul	xl
weight	3	1	3	1	**1**	2	4	1	**3**
variable	yl	a	b	c	d	e	f	g	h
weight	1	1	1	1	1	1	1	1	1

After Fanout Reduction and Variable Merging						
variable	(u_3,yl,d)	y_2	(x_2**,**xl)	(a,f,h)	b	(c,g,ul)
weight	5	4	**5**	3	1	3

Table 2: The weights of the variables.

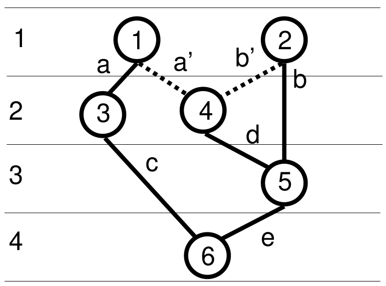

Figure 10: An example illustrating the fanout reduction and variable merging.

```
Update_LB_on_Reg()
{
 loop:
        Reg_num = LB_Reg;
        Adjust_Time_Frames_with_Register_Constraint(Reg_num);
        if there is an operation O_k such that ASAP_k > ALAP_k,
                increment LB_Reg and go to loop;
        calculate new lower bound LB'_Reg using the temporary time frames;
        if LB'_Reg > Reg_num, increment LB_Reg and go to loop;
}

Adjust_Time_Frames_with_Register_Constraint(Reg_num)
{
        For each variable V_{i,j},
                Z = [1, ASAP_i];
                Schedule the source O_i temporarily to ASAP_i and then
                        temporarily adjust the ALAP csteps of its predecessors accordingly;
                LB'_Reg = ⌈ (Σ_{V_{i,j}} M_{i,j,z}) / |Z| ⌉;
                Resume all the time frames;
                if(LB'_Reg > Reg_num), increment ASAP_i and ASAP'_i by one;

                Z = [ALAP'_j, total cstep number + 1];
                Schedule the destination O_j temporarily to ALAP_j and then
                        temporarily adjust the ASAP csteps of its successors accordingly;
                LB'_Reg = ⌈ (Σ_{V_{i,j}} M_{i,j,z}) / |Z| ⌉;
                Resume all the time frames;
                if(LB'_Reg > Reg_num), decrement ALAP_j and ALAP'_j by one;
}
```

Figure 11: Algorithm for refining the register estimate.

Estimate_Bus_Cost()

{

$\Gamma = \{V_{i,j} \mid O_i \ and \ O_j \ CANNOT \ be \ chained\}$;

for each cstep interval $Z \subseteq [1, total \ cstep \ number]$,

$\Psi_I(Z) = \{V_{i,j} \mid V_{i,j} \in \Gamma \ and \ [ASAP_j, ALAP_j] \subseteq Z\}$;

$\Psi_O(Z) = \{V_{i,j} \mid V_{i,j} \in \Gamma \ and \ [ASAP_i', ALAP_i'] \subseteq Z\}$;

for each two variable $V_{i,j}$ and $V_{i,k}$ in $\Psi_I(Z)$,

if $(ASAP_j = ALAP_k \ and \ ALAP_j = ASAP_k)$,

remove $V_{i,k}$ from $\Psi_I(Z)$;

$LB_{I\_Bus} = max_z \ (\lceil \ |\Psi_I(Z)| \ / \ |Z| \ \rceil)$;

$LB_{O\_Bus} = max_z \ (\lceil \ |\Psi_O(Z)| \ / \ |Z| \ \rceil)$;

if(Bus is bi-directional) $LB_{Bus} = max \ (LB_{I\_Bus}, LB_{O\_Bus})$;

else $LB_{Bus} = LB_{I\_Bus} + LB_{O\_Bus}$;

$Est\_Bus\_Cost = LB_{Bus} \times Area_{Bus}$;

update Est_Bus_Cost;

return(Est_Bus_Cost);

}

Figure 12: Our bus cost estimation algorithm.

Update_LB_on_Bus()

{

loop1: /* I_Bus Refinement */

$I\_Bus\_num = LB_{I\_Bus}$;

Adjust_Time_Frames_with_Bus_Constraint(I_BUS, I_bus_num);

if there is an operation O_k such that $ASAP_k > ALAP_k$,

increment LB_{I_Bus} and go to loop1;

calculate new lower bound LB'_{I_Bus} using the temporary time frames;

if $LB'_{I\_Bus} > I\_Bus\_num$, increment LB_{I_Bus} and go to loop1;

loop2: /* O_Bus Refinement */

$O\_Bus\_num = LB_{O\_Bus}$;

Adjust_Time_Frames_with_Bus_Constraint(O_BUS, O_bus_num);

if there is an operation O_k such that $ASAP_k > ALAP_k$,

increment LB_{O_Bus} and go to loop2;

calculate new lower bound LB'_{O_Bus} using the temporary time frames;

if $LB'_{O\_Bus} > O\_Bus\_num$, increment LB_{O_Bus} and go to loop2;

}

Adjust_Time_Frames_with_Bus_Constraint(MODE, Bus_num)

{

For each operation O_i,

$Z = [1, ASAP_i]$;

Schedule the operation O_i temporarily to $ASAP_i$ and then

temporarily adjust the ALAP csteps of its predecessors accordingly;

if(MODE = I_BUS) $LB'_{Bus} = \lceil \Psi_I(Z) \ / \ |Z| \rceil$;

else $LB'_{Bus} = \lceil \Psi_O(Z) \ / \ |Z| \rceil$;

Resume all the time frames;

if($LB'_{Bus} > Bus\_num$), increment $ASAP_i$ and $ASAP'_i$ by one;

$Z = [ALAP'_i, total\ cstep\ number + 1]$;

Schedule the operation O_i temporarily to $ALAP_i$ and then

temporarily adjust the ASAP csteps of its successors accordingly;

if(MODE = I_BUS) $LB'_{Bus} = \lceil \Psi_I(Z) \ / \ |Z| \rceil$;

else $LB'_{Bus} = \lceil \Psi_O(Z) \ / \ |Z| \rceil$;

Resume all the time frames;

if($LB'_{Bus} > Bus\_num$), decrement $ALAP_i$ and $ALAP'_i$ by one;

}

Figure 13: Algorithm for refining the bus estimates.

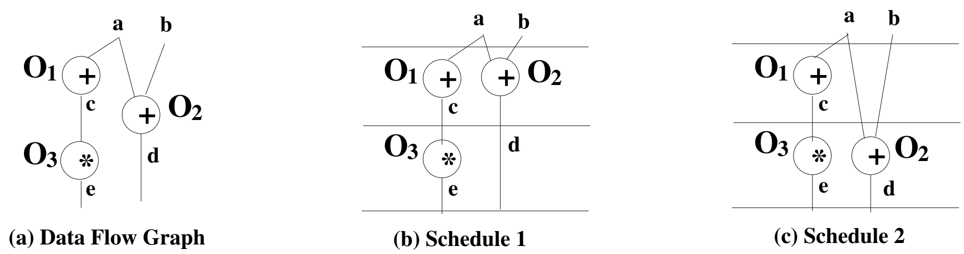

Figure 14: Dependency between the different types of resources

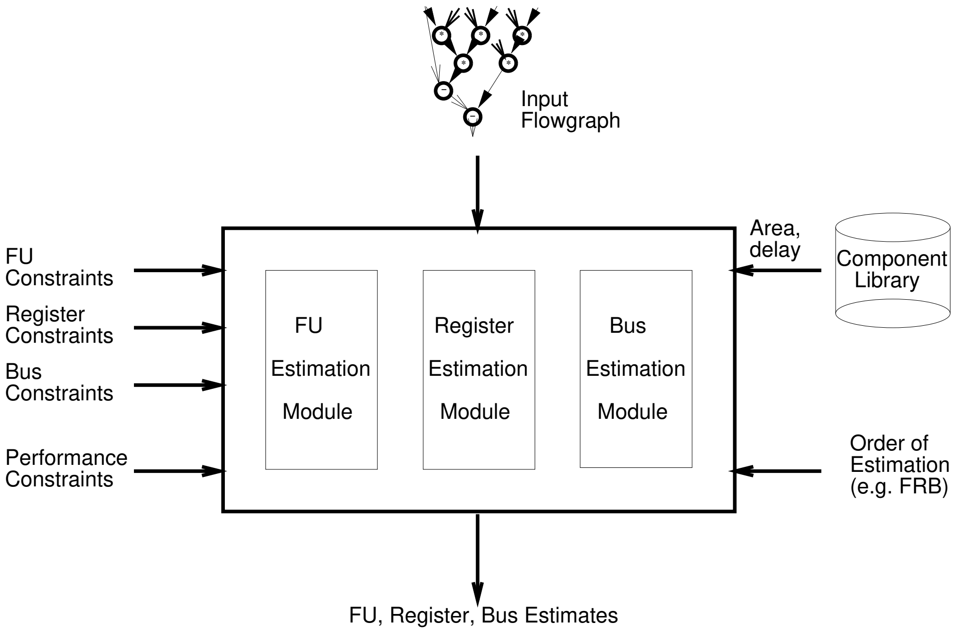

Figure 15: The Structure of the estimation system

example	performance (ns)	Without Refinement		With Refinement	
		Resource Counts†	CPU Time‡	Resource Counts†	CPU Time‡
Differential Equation	120	1+,3∗,4r,3/2b	0.01	2+,3∗,5r,3/2b	0.11
	140	1+,2∗,3r,2/2b	0.03	2+,2∗,4r,2/2b	0.23
EWF	340	3+,3∗,11r,5/3b	0.08	3+,3∗,11r,6/4b	0.80
	360	2+,2∗,9r,4/3b	0.10	2+,2∗,10r,4/3b	7.73
	380	2+,2∗,9r,4/3b	0.10	2+,2∗,9r,4/3b	7.80
	400	2+,2∗,8r,4/2b	0.13	2+,2∗,9r,4/2b	12.80
	420	2+,1∗,8r,3/2b	0.13	2+,1∗,8r,3/2b	29.75
ARF	120	4+,4∗,6r,4/4b	0.02	4+,4∗,6r,6/4b	0.10
	140	2+,3∗,4r,4/4b	0.01	4+,4∗,4r,4/4b	0.56
	160	2+,3∗,3r,3/3b	0.02	2+,3∗,4r,4/3b	0.74
	180	2+,2∗,3r,3/3b	0.04	2+,2∗,3r,3/3b	0.94
DCT	140	6+,7∗,11r,8/7b	0.03	6+,8∗,13r,8/8b	2.65
	160	4+,6∗,8r,6/6b	0.06	5+,6∗,9r,8/6b	4.37
	180	4+,5∗,8r,6/6b	0.04	4+,5∗,8r,6/6b	5.87
Jacobian Example	660	11+,27∗,9m,31r,25/23b	14.90	11+,27∗,9m,31r,25/23b	45.27
	680	10+,26∗,8m,30r,24/23b	17.82	10+,26∗,8m,30r,24/23b	67.20

- †: Resources are estimated separately.
- ‡: CPU times are accumulated for separate estimation steps
- bus count : Input/Output bus counts

Table 3: Effects of Refinement Steps

ordering of estimation	performance (ns)	
	120	140
FRB	2a,3$*$,6r,4/2b	2a,2$*$,5r,4/2b
FBR	2a,3$*$,6r,4/2b	2a,2$*$,5r,4/2b
RFB	2a,3$*$,5r,4/2b	2a,2$*$,4r,4/2b
RBF	1a,4$*$,5r,3/2b	2a,2$*$,4r,2/2b
BFR	1a,4$*$,7r,3/2b	2a,2$*$,5r,2/2b
BRF	1a,4$*$,7r,3/2b	2a,2$*$,5r,2/2b

- a: ALU (delay = 15 ns)
- $*$: multiplier (delay = 24.4 ns)
- clock cycle = 20 ns

Table 4: Differential Equation

ordering of estimation	performance (ns)				
	340	360	380	400	420
FRB	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,9r,4/3b
FBR	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,10r,4/2b
RFB	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,8r,4/2b
RBF	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,8r,3/2b
BFR	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,9r,3/2b
BRF	3+,3∗,11r,6/4b	2+,2∗,10r,4/3b	2+,2∗,9r,4/3b	2+,2∗,9r,4/2b	2+,1∗,9r,3/2b

- $+$: adder (delay = 15 ns), $*$: multiplier (delay = 24.4 ns), clock cycle = 20 ns

Table 5: Elliptic Wave Filter

ordering of estimation	performance (ns)			
	120	140	160	180
FRB	4+,4$*$,6r,6/5b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/4b	2+,2$*$,3r,3/3b
FBR	4+,4$*$,6r,6/5b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/4b	2+,2$*$,4r,3/3b
RFB	4+,4$*$,6r,6/5b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/4b	2+,2$*$,3r,3/3b
RBF	6+,4$*$,6r,6/4b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/3b	2+,2$*$,3r,3/3b
BFR	6+,4$*$,8r,6/4b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/3b	2+,2$*$,4r,3/3b
BRF	6+,4$*$,8r,6/4b	4+,4$*$,4r,4/4b	2+,3$*$,4r,4/3b	2+,2$*$,3r,3/3b

- $+$: adder (delay = 7.5 ns), $*$: multiplier (delay = 15 ns)
- clock cycle = 20ns

Table 6: AR Filter

ordering of estimation	performance (ns)		
	140	160	180
FRB	6a,8∗,14r,11/8b	5a,6∗,10r,8/6b	4a,5∗,8r,6/6b
FBR	6a,8∗,14r,11/8b	5a,6∗,10r,8/6b	4a,5∗,10r,6/6b
RFB	6a,8∗,13r,11/8b	5a,6∗,9r,8/6b	4a,5∗,8r,6/6b
RBF	6a,8∗,13r,8/8b	5a,6∗,9r,8/6b	4a,5∗,8r,6/6b
BFR	6a,8∗,14r,8/8b	5a,6∗,10r,8/6b	4a,5∗,11r,6/6b
BRF	6a,8∗,13r,8/8b	5a,6∗,9r,8/6b	4a,5∗,10r,6/6b

- a: ALU (delay = 15 ns), ∗: multiplier (delay = 24.4 ns)
- clock cycle = 20ns

Table 7: Discrete Cosine Transformer

ordering of estimation	performance (ns)	
	660	680
FRB	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b
FBR	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b
RFB	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b
RBF	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b
BFR	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b
BRF	11a,27$*$,9m,31r,25/23b	10a,26$*$,8m,30r,24/23b

- a: ALU (delay = 15 ns), $*$: multiplier (delay = 24.4 ns)
- m: memory access (delay = 50 ns), clock cycle = 20 ns

Table 8: Jacobian Transformer

delay (ns)	Actual Designs						Our Estimation(FRB)		
	HAL[17]			Hand-Optimized					
	FU	reg	bus	FU	reg	bus	FU	reg	bus
340	3+,3$*$	-	-	3+,3$*$	11	-	3+,3$*$	11	10
360	3+,2$*$	-	-	2+,2$*$	10	-	2+,2$*$	10	7
380	2+,2$*$	12	-	2+,2$*$	9	-	2+,2$*$	9	7
400	2+,2$*$	-	-	2+,2$*$	9	-	2+,2$*$	9	6
420	2+,1$*$	12	-	2+,1$*$	9	-	2+,1$*$	9	7

- +: adder (delay = 15 ns), $*$: multiplier (delay = 24.4 ns)
- clock cycle = 20 ns
- reg: Input/output values are stored in registers.
- bus: Uni-directional buses are used.

Table 9: Comparison with actual designs for EWF example - I

delay (ns)	Actual Designs						Our Estimation(FRB)		
	HAL[17]			OASIC[12]					
	FU	reg	bus	FU	reg	bus	FU	reg	bus
340	3+,2$\star$	12	-	3+,2$\star$	10	10	3+,2$\star$	10	10
360	3+,1$\star$	-	-	3+,1$\star$	10	9	3+,1$\star$	10	8
380	2+,1$\star$	12	-	2+,1$\star$	9	7	2+,1$\star$	9	7

- +: adder (delay = 15 ns), $\star$: pipelined multiplier (delay = 24.4 ns)
- clock cycle = 20 ns
- reg: Input/output values are stored in registers.
- bus: Uni-directional buses are used.

Table 10: Comparison with actual designs for EWF example - II

delay (ns)	Actual Designs						Our Estimation(FRB)		
	InSyn [18]			ALPS[19]					
	FU	reg	bus	FU	reg	bus	FU	reg	bus
340	$3+,3*$	8	7	$3+,3*$	-	6	$3+,3*$	8	6
	$3+,2\star$	8	7	$3+,2\star$	-	6	$3+,2\star$	7	6
360	$2+,2*$	8	5	$2+,2*$	-	6	$2+,2*$	6	4
	$3+,1\star$	-	-	$3+,1\star$	-	6	$3+,1\star$	6	4
380	$2+,2*$	8	4	$2+,2*$	-	-	$2+,2*$	6	3
	$2+,1\star$	8	4	$2+,1\star$	-	6	$2+,1\star$	5	3
400	$2+,2*$	-	-	$2+,2*$	-	4	$2+,2*$	5	3
	$2+,1\star$	-	-	$2+,1\star$	-	4	$2+,1\star$	5	3
420	$2+,1*$	-	-	$2+,1*$	-	4	$2+,1*$	6	3
	$2+,1\star$	-	-	$2+,1\star$	-	-	$2+,1\star$	4	3

- $+$: adder (delay = 15 ns), $*$: multiplier (delay = 24.4 ns)
- $\star$: pipelined multiplier (delay = 24.4 ns)
- clock cycle = 20 ns
- reg: Input/output values are not stored in registers.
- bus: Bi-directional buses are used.

Table 11: Comparison with actual designs for EWF example - III

delay (ns)	Actual Design Hand-Optimized			Our Estimation(FRB)		
	FU	reg	bus	FU	reg	bus
120	4+,4∗	6	-	4+,4∗	6	6
140	4+,4∗	6	-	4+,4∗	4	4
160	2+,3∗	6	-	2+,3∗	5	4
180	2+,2∗	5	-	2+,2∗	3	3

- $+$: ALU (delay = 7.5 ns)
- $*$: multiplier (delay = 15 ns)
- clock cycle = 20 ns
- reg: Input/output values are not stored in registers.
- bus: Bi-directional buses are used.

Table 12: Comparison with actual design for AR filter example

delay (ns)	Actual Design ILP[20]			Our Estimation(FRB)		
	FU	reg	bus	FU	reg	bus
140	6+,5$\star$	12	-	6+,5$\star$	11	11
160	5+,4$\star$	12	-	5+,4$\star$	9	7
180	4+,3$\star$	12	-	4+,3$\star$	7	6

- $+$: ALU (delay = 15 ns)
- $\star$: pipelined multiplier (delay = 24.4 ns)
- reg: Input/output values are not stored in registers.
- bus: Bi-directional buses are used.

Table 13: Comparison with actual designs for DCT example - I

delay (ns)	Actual Design SALSA[22]			Our Estimation(FRB)		
	FU	reg	bus	FU	reg	bus
200	4+,3$\star$	12	-	4+,3$\star$	8	5
220	4+,2$\star$	13	-	3+,2$\star$	8	5
260	3+,2$\star$	14	-	3+,2$\star$	8	4
380	3+,1$\star$	14	-	2+,1$\star$	8	3

- +: ALU (delay = 15 ns)
- $\star$: pipelined multiplier (delay = 24.4 ns)
- reg: Input/output values are stored in registers.
- bus: Bi-directional buses are used.

Table 14: Comparison with actual designs for DCT example - II

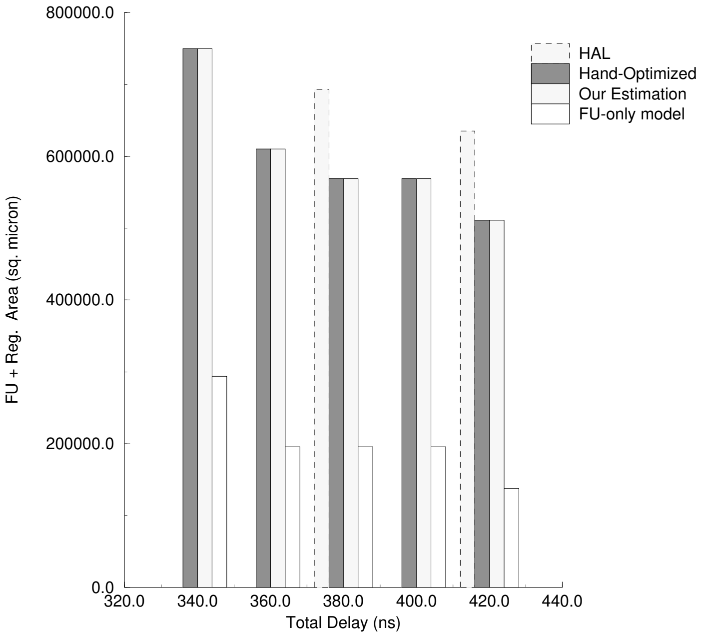

Figure 16: Comparison with actual designs for EWF example - I (Table 9)

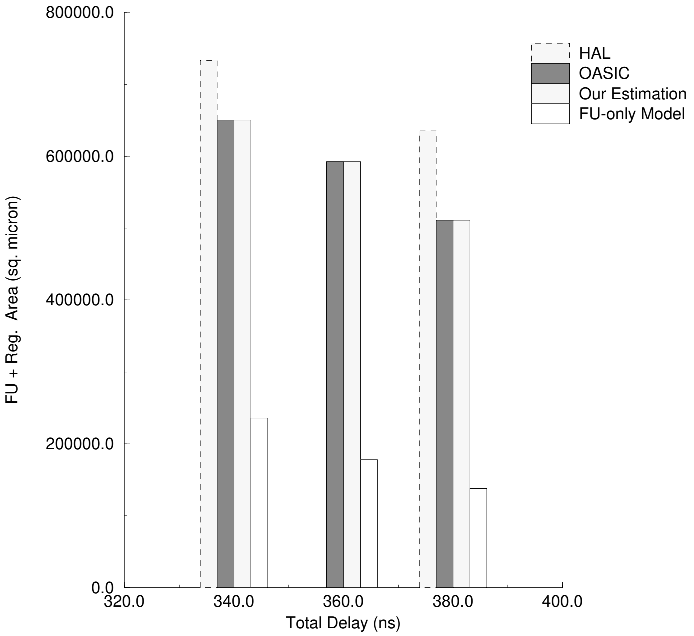

Figure 17: Comparison with actual designs for EWF example - II (Table 10)

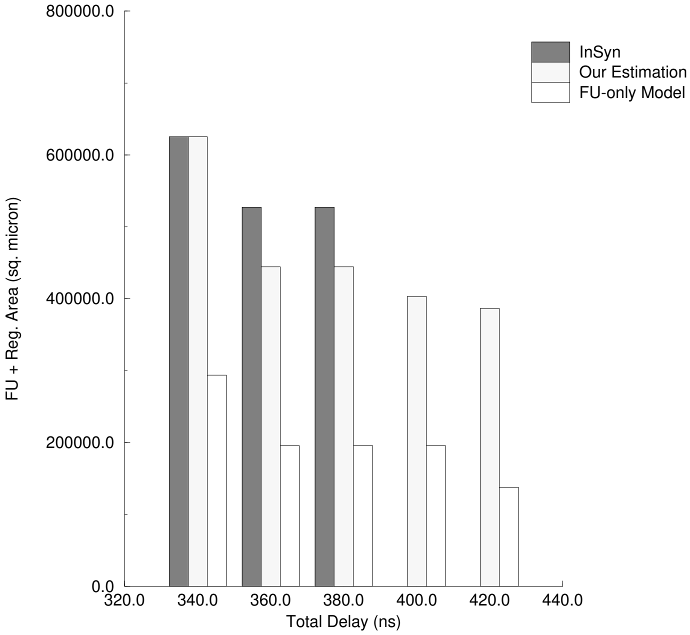

Figure 18: Comparison with actual designs for EWF example - III (Table 11)

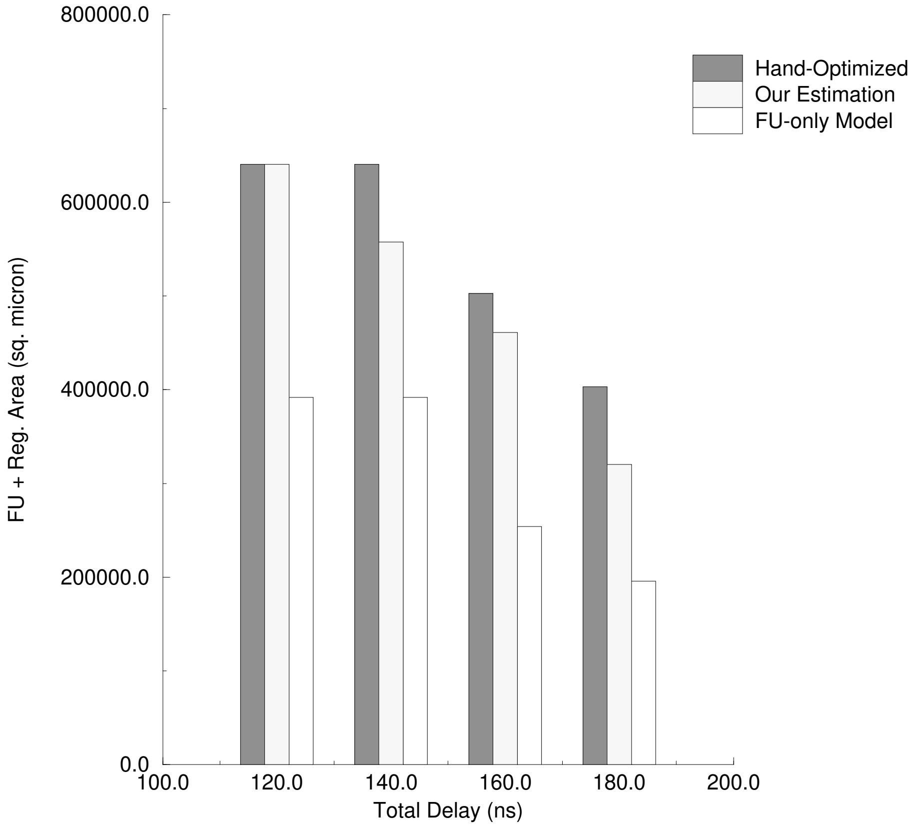

Figure 19: Comparison with actual designs for AR filter (Table 12)

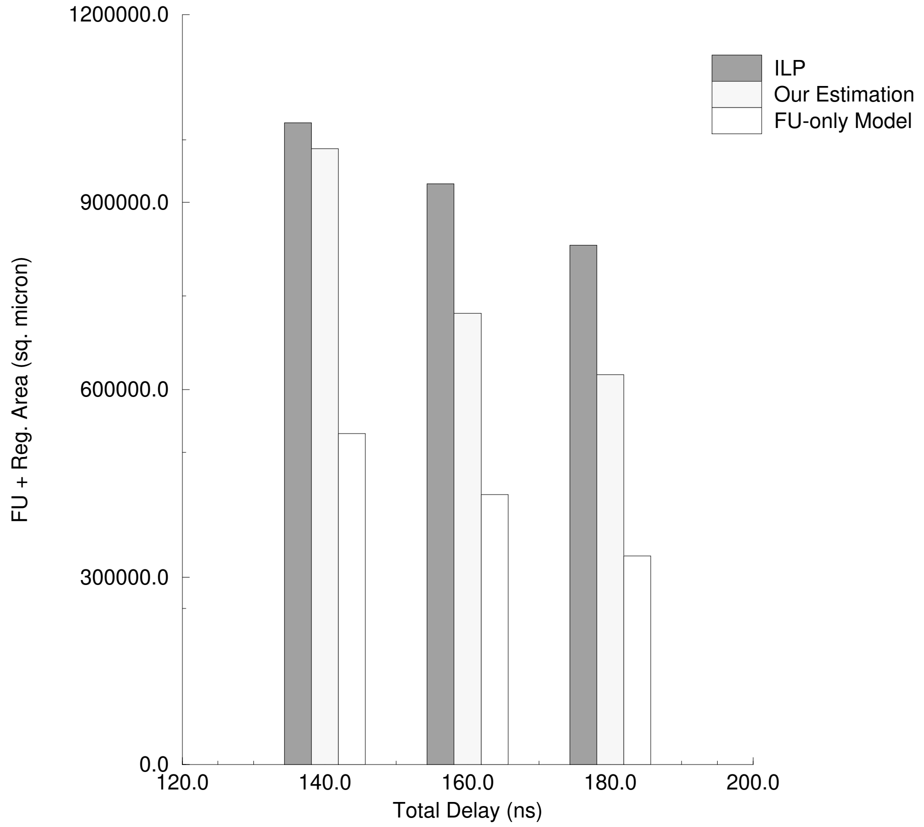

Figure 20: Comparison with actual designs for DCT example - I (Table 13)

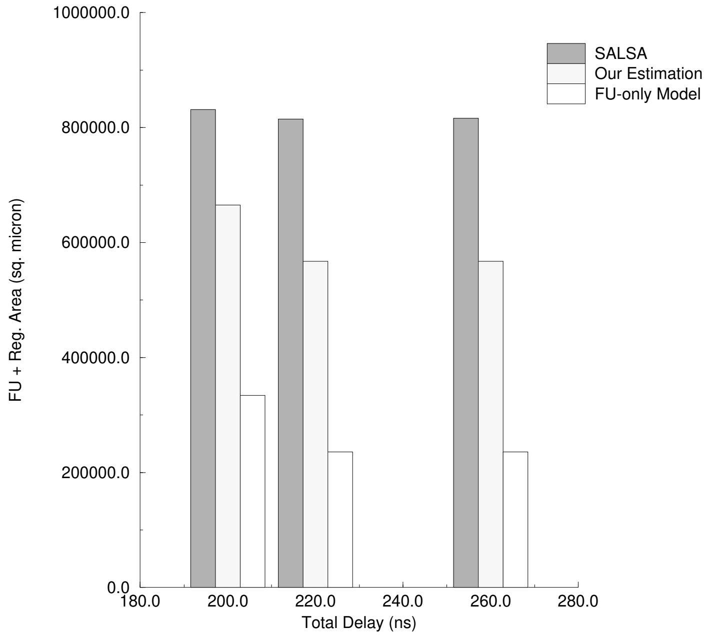

Figure 21: Comparison with actual designs for DCT example - II (Table 14)

Footnotes, affiliation, and acknowledgement

Manuscript received ____________________

Affiliations of Authors:

S. Y. Ohm was with the Electrical & Computer Engineering Department, University of California, Irvine. He is currently with the Department of Computer Engineering, Seoul Women's University, KOREA.

F. J. Kurdahi is with the Electrical & Computer Engineering Department, University of California, Irvine, CA 92717

N. D. Dutt is with the Information & Computer Science Department, University of California, Irvine, CA 92717

This work was supported by a MICRO grant from the University of California and Compass Design Automation Inc., and by a Fellowship from the Korea Organization of Science and Engineering Foundation. A preliminary version of this work appeared in the Proceedings of ICCAD-94 under the title: "Comprehensive Lower Bound Estimation from Behavioral Specifications".

Footnote 1: We say that the time frame of operation O_i is *valid* only if $ASAP_i \leq ALAP_i$.

Footnote 2: We did not include bus cost in the total area cost since it is not easy to quantify the area of buses before physical level synthesis.

Seong Yong Ohm received the B.E., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University, Seoul, Korea in 1985, 1987, and 1992 respectively.

From 1992 to 1993, he was working with Research Institute of Advanced Computer Technology in Seoul as a senior researcher. From 1993 to 1995, he was working with the Department of Electrical & Computer Engineering at the University of California, Irvine as a post-doctorial researcher. Since 1996, he has been with the Department of Computer Engineering at Seoul Women's University, Seoul, Korea, where he is currently an Assistant Professor. He received three fellowships from Korea Organization of Science and Engineering Foundations in 1985, 1987, and 1993. His areas of interest are high-level synthesis of digital circuits, VLSI system modeling and estimation, hardware/software co-design, design automation, and computer graphics.

Fadi J. Kurdahi (S'85 - M'87) received the Bachelor of Engineering degree in Electrical Engineering from the American University of Beirut, Lebanon in 1981. He received the M.S. degree in Electrical Engineering and the Ph.D. degree in Computer Engineering from the University of Southern California, Los Angeles, CA, in 1982 and 1987, respectively.

Since 1987, he has been with the Department of Electrical & Computer Engineering at the University of California, Irvine, where he is currently an Associate Professor. He received an NSF Research Initiation Award in 1989, and two ACM/SIGDA fellowships in 1991 and 1992. His areas of interest are high-level synthesis of digital circuits, VLSI systems design and layout, and design automation. He has served on program committees of ISSS, ED$TC, ISSS, ISCAS. Dr. Kurdahi was Associate Editor of IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 1993-1995.

He is a member of IEEE and ACM.

Nikil D. Dutt (S'83-M'89-SM'96) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1989. He is currently an Associate Professor of ICS and ECE at the University of California at Irvine. His research interests include high-level and system synthesis, system specification languages, and hardware/software issues in embedded system design. He is a co-author of the book "High-Level Synthesis: Introduction to Chip and System Design" (Norwell, MA: Kluwer Academic, 1992). He received the Best Paper Award at two consecutive Conferences on Hardware Description Languages (CHDL89 and CHDL91). He has served on several conference program committees, including ICCAD, EDT&C, EURODAC, ISSS and CHDL. He is a member of the IEEE, the ACM and the IFIP WG 10.5.