A Comprehensive Estimation Technique for High-Level Synthesis *

Seong Y. Ohm†, Fadi J. Kurdahi†, Nikil Dutt‡, and Min Xu‡

† Department of Electrical & Computer Engineering
‡ Department of Information & Computer Science
University of California, Irvine, CA 92717

Abstract

We present an integrated approach aimed at predicting layout area needed to implement a behavioral description for a given performance goal. Our approach is novel because: (1) it accounts for all types of RT level components (FUs, buses, registers), (2) it is highly flexible, allowing the designer to tradeoff one type of resources with another, and considers dependencies between these different types, (3) it is vertically integrated to include provably accurate physical level estimators, and hence provides realistic accounting of layout effects, and (4) it uses a timing model with finer granularity, accounting for various delays in RTL datapaths. We demonstrate our technique on a variety of HLS benchmarks and show that efficient and effective design space exploration can be accomplished using this technique.

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 for guidance in making design decisions. Specifically, we need to have the capability of *bounding* the design space prior to starting the HLS tasks.

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, interconnect and other layout considerations can indeed dominate the cost function.

Our approach is novel and differs from previous approaches because it contains the following features:

1. At the RT level, our cost model accounts for registers and buses as well as FUs.
2. Our approach is *horizontally* integrated and highly flexible because we consider the dependencies between the different types of resources as well as the ordering in which the resources are allocated.
3. Our approach is also *vertically* integrated, since our estimation is extended to the physical level. By linking our RT level estimates to provably accurate physical level estimators at both component level and chip level [1], it is possible to account for layout effect on both components (e.g. variations in area and delay with component shape and aspect ratio), and complete RT level designs (e.g. wiring effects, unused area).
4. Our timing model uses a finer granularity that permits the modeling of FU, register, and interconnect delays. Our approach accommodates all of these factors in estimation along with transfer delays among these hardware resources.

We have developed efficient algorithms and heuristics to support this model. Our initial experiments on some HLS benchmarks [2] 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.

*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.

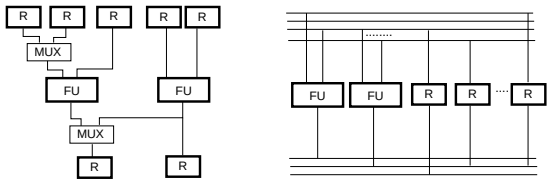


Figure 1: Architectural models: (a) multiplexor-based architecture and (b) bus-based architecture.

2 Previous Work

BUD [3], an earlier works in linking behavior with physical level design, clearly indicated the significance of interconnect and other layout effects -traditionally considered as second order in HLS- on the overall implementation area and delay. This approach, however, constrains the designer to explore a fraction of the design space. Furthermore, the accuracy of the estimators was not reported.

There is some recent work on estimating lower bounds on area cost and total control steps (or csteps). [4], [5], [6], [7], and [8] are mainly concerned with FUs in their area cost models. [9], [10], [11], [12], and [13] use more elaborate cost function for estimation. However, they do not consider both: (1) dependencies among different types of resources, and (2) linkage to physical level. By contrast, our approach considers both aspects.

3 Architectural Model and Problem Definition

We consider two styles of target architectures as shown in Figure 1: multiplexor-based and bus-based. In this paper, however, we confine our scope to bus-based architectural model (Figure 1(b)). We also assume that only registers are used for storage resources. We do not take into account single- or multi-port memories at this time. 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.

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 delays, and interconnect delays are initially estimated by the user or selected from a library. Given these information, we estimate lower bounds on the number of FUs of each type, the number of registers, and the number of buses (bi-directional or uni-directional).

4 Our Approach

We describe an integrated approach which allows the estimation process to begin from the *behavioral* level (as opposed to RT level). This approach is illustrated in Figure 2. For a given behavioral specifications, our system generates *lower bound* estimates on resources needed in order to achieve a user-defined performance goal. This in turn is used to obtain an approximate topology of the resulting physical layout.

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 and register estimation in prior work [13, 9, 6], in this work we extend this approach for bus estimation as well.

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. 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.

Figure 3 shows the overall structure of the area cost estimation algorithm **LBE**, when FU cost is first estimated, register cost next, and bus cost finally. As shown later in this paper, we can accommodate different orderings of these steps and account for the conditional lower bounds.

The FU and register area cost estimation techniques were published in [13]. In this paper, we concentrate on estimating bus area cost and linking these estimates to the physical (layout) level. In order to accomplish that, we need to do the following:

1. estimate the number and type of buses needed to ac-

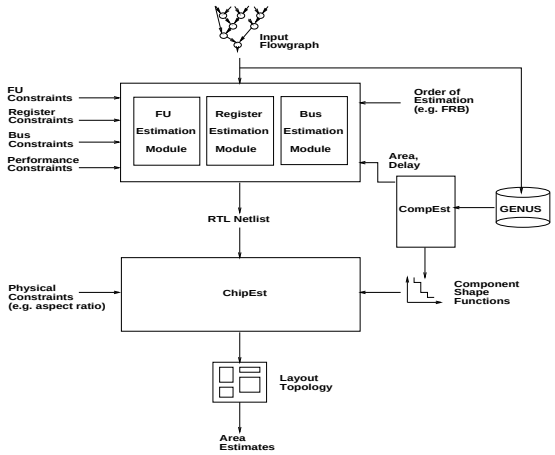


Figure 2: The structure of the estimation system.

```
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 3: One formulation of our area cost estimation algorithm.

commodate the needed data transfers, and

2. estimate the physical design topology based on the FU, register, and bus estimates.

5 Bus Estimation

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 4 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 the 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.

```
Estimate_Bus_Cost()
{
        Γ = {V_i,j | O_i and O_j CANNOT be chained};
        for each cstep interval Z ⊆ [1, total cstep number],
            Ψ_I(Z) = {V_i,j | V_i,j ∈ Γ and [ASAP_j, ALAP_j] ⊆ Z};
            Ψ_O(Z) = {V_i,j | V_i,j ∈ Γ and [ASAP'_i, ALAP'_i] ⊆ Z};
            for each two variable V_i,j and V_i,k in Ψ_I(Z),
                if (ASAP_j = ALAP_k and ALAP_j = ASAP_k),
                    remove V_i,k from Ψ_I(Z);
        LB_I_Bus = max_z (⌈ |Ψ_I(Z)| / |Z| ⌉);
        LB_O_Bus = max_z (⌈ |Ψ_O(Z)| / |Z| ⌉);
        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 × Area_Bus;
        update Est_Bus_Cost;
        return(Est_Bus_Cost);
}
```

Figure 4: Our bus cost estimation algorithm.

The initial lower bounds on bus counts can be refined further to obtain tighter lower bounds. The details of the refinement technique are described in [20]. Similar algorithms are used for refining the lower bounds on FU counts and on register count respectively as described in [13]. The total time complexity of this algorithm is $O(C^2 \cdot (E + N^2) + N \cdot E)$, where C is the total number of csteps, E the number of values, and N the number of operations.

6 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 5(a). For this example, there exist only 2 possible schedules shown in Figure 5(b) and 5(c) respectively, if the total cstep number is 2. The schedule in Figure 5(b) requires 2 adders, 1 multiplier, and 2 registers, whereas that in Figure 5(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

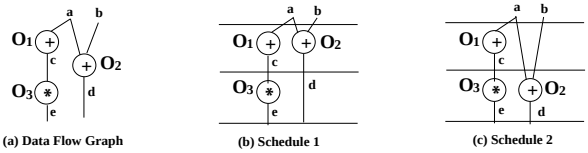


Figure 5: Dependency between the different types of resources.

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.

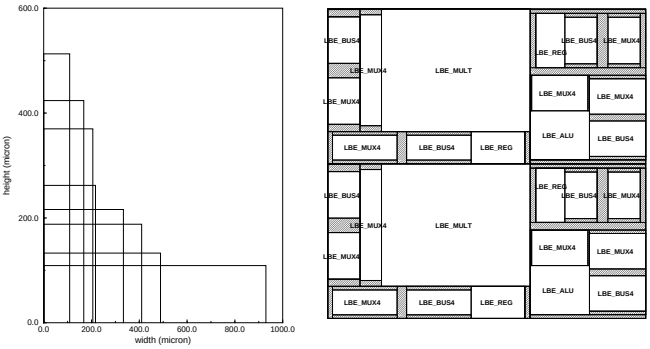
Consider the example in Figure 5 again, assuming that the FU cost estimation is followed by register cost estimation. When we estimate the register cost for that example, we assume that 1 adder and 1 multiplier are used, since their lower bounds estimated in the FU cost estimation step are 1 each. This 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.

7 Physical Level Estimation

Once we have obtained estimates of FU, register, and bus costs, we can generate an RT level netlist in VHDL based on the architectural model shown in Figure 1. Given the netlist specification, we use two physical level estimation tools[1]: CompEst and ChipEst to obtain an approximate topology of the layout. CompEst is a component estimation tool which predicts the area and delay of a given RT level component netlist. While CompEst can handle different design styles, we currently assume that components are implemented in standard cells for three reasons: (1) standard cells can implement arbitrary components using component generators such as GENUS [21], (2) standard cells provide flexibility in generating alternative physical implementations which can result in highly efficient floorplans of the overall design, and (3) this methodology is used by several leading tool vendors such as Compass and others to implement RT level designs.

Given a specification of a particular component, we first use GENUS to generate a set of Boolean equations to implement the required functionality. Next, we use CompEst to predict the shape function for each component. CompEst predicts the effects of some logic synthesis tasks such as technology mapping as well as the effects of physical design. This shape function can be obtained by estimating the dimensions of a standard cells block with varying number of rows. An example shape function for an ALU is shown in Figure 6(a). Additionally, CompEst estimates the critical path delay of each configuration taking into account wiring delay as well as false paths. Benchmarking has shown that CompEst can estimate area with about 5% accuracy and static delay with about 7-10% accuracy [1].

Once we have obtained a shape function for each component, we use ChipEst to generate an approximate topology of the overall design. ChipEst employs a partial slicing technique to generate a highly efficient approximate topology of the design, and choose the most appropriate implementation of each component. Experience has shown that component area and delay do vary (and sometimes significantly) with aspect ratio [17]. Thus this step provides valuable feedback to the designer by comparing the resulting estimated component area and delay figures to the ones assumed initially.



(a) (b)

Figure 6: (a) Shape function for an ALU (b) An example output of ChipEst.

If large differences are observed, then the user may have to go back and modify his/her assumptions accordingly. An example output produced ChipEst is shown in Figure 6(b).

Since at this point no particular binding is assumed, we cannot estimate the overall system delay or clock cycle since that would depend heavily on connectivity, which has not been determined yet. In a bottom up design methodology, the designer can use the estimated topology as a guidance to the design process and attempt to minimize the contribution of wiring by properly binding components to abstract operations, values, and transfers. Such approaches have been shown to produce good results [18].

8 Implementation and Experiments

We implemented our integrated estimation approach in the C language on a SUN Sparcstation. The structure of the system is shown in Figure 2. In the first phase, 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. The user can additionally control individual resource counts and run the system to estimate the remaining resources. A library of components provide the area and delay data for each component. The main components used in this library are shown in Table 1. The area and delay figures were estimated using netlists generated by GENUS and estimated by CompEst. Components are assumed to have a square aspect ratio in this phase only. This first phase produces a fully-connected netlist in VHDL. This VHDL netlist is then passed on to the physical level estimation tools. In this phase, the full shape function of each component is generated by CompEst and used by ChipEst to predict the layout topology of the whole design. If the user specifies an aspect ratio constraint, then the system will report one alternative which most closely matches the desired aspect ratio. Otherwise, a shape function is produced.

In order to validate our estimation system, we applied it

Table 1: GENUS/VTI 0.8μ standard cell component library.

Unit (4 bit)	Area (sq. micron)	Delay (ns)	Aspect Ratio
Multiplier	386259	146.4	1:1
ALU	76220	59.8	1:1
Register	27244	20.1	1:1

Table 2: Comparison with actual designs for EWF example.

Schedule Length	Actual Designs						Our Estimation (FRB)		
	InSyn [14]			ALPS[15]					
	FU	reg	bus	FU	reg	bus	FU	reg	bus
17	3+,3∗	8	7	3+,3∗	-	6	3+,3∗	8	6
	3+,2⋆	8	7	3+,2⋆	-	6	3+,2⋆	7	6
18	2+,2∗	8	5	2+,2∗	-	6	2+,2∗	6	4
	3+,1⋆	-	-	3+,1⋆	-	6	3+,1⋆	6	4
19	2+,2∗	8	4	2+,2∗	-	-	2+,2∗	6	3
	2+,1⋆	8	4	2+,1⋆	-	6	2+,1⋆	5	3
20	2+,2∗	-	-	2+,2∗	-	4	2+,2∗	5	3
	2+,1⋆	-	-	2+,1⋆	-	4	2+,1⋆	5	3
21	2+,1∗	-	-	2+,1∗	-	4	2+,1∗	6	3
	2+,1⋆	-	-	2+,1⋆	-	-	2+,1⋆	4	3

- +: adder, ∗: multiplier(2 cycle), ⋆: 2-stage pipelined multiplier
- reg: Input/output values are not stored in registers.
- bus: Bi-directional buses are used.

to some well-known High-Level Synthesis Benchmarks[2], including *the AR Filter* (ARF), *the 5th order Elliptic Wave Filter* (EWF), and *Discrete Cosine Transformer* (DCT). We also tried our estimation on a large example, the *Jacobian Transformer* [19].

– Experiment 1: 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 several existing systems including OASIC [10], InSyn [14], ALPS [15], and ILP [16]. In Table 2 we show our comparison results for one particular experiment using the EWF example. 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. More experimental results are presented in [20]. Overall, these experimental results clearly show that our estimates are quite accurate. Indeed, the FU count is exactly the same as the optimal results generated by both systems. Furthermore, in most cases, the lower bound on register (bus) cost is 1 or 2 units below the actual register (bus) count. We have done many additional extensive experimental results on FU and register counts for other benchmarks which are reported in [20]. In this paper, we concentrate on the bus and overall area estimation and report on comparisons with the EWF since this was the only published example for which we could find designs reporting resource counts for FUs, register, and buses.

– Experiment 2: Exploring the design space

In the second set of experiments, we used the estimation system shown in Figure 2 to explore the design space of the HLS benchmarks above. The first parameter in the exploration is the estimation order. 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 setup an initial constraints on Total Execution Time (TET). Next, we ran the system six times each with a different estimation order. This is repeated for different constraints on execution time.

Table 3 summarizes the estimation results for three benchmarks: ARF, EWF, and DCT. CPU times for these experiments were between 10 and 250 seconds. In this table, 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. Notation "x/yb" denotes the lower bound on number of input buses is x while that of output buses is y. In these experiments, we assumed that all input and output data values are considered to be stored in registers.

Table 3: Summary of Experimental Results.

Ex.	Total Execution Time (TET) (ns)	Estimation Order	RTL Estimation # of Components	ChipEst Area (μ^2)
ARF	1680	all	4a 4∗ 6r 4/4b	3713005
	1960	FRB,FBR,BFR	8a 3∗ 5r 4/4b	3697920
		RFB,RBF,BRF	4a 4∗ 5r 4/4b	3609600
	2240	FRB,FBR,BFR,BRF	2a 3∗ 5r 4/4b	2579202
		RFB,RBF	2a 3∗ 4r 4/4b	2382120
	2520	all	2a 2∗ 4r 3/3b	1928960
	2800	all	2a 2∗ 4r 3/3b	1928960
EWF	2520	all	4a 2∗ 10r 6/4b	5227530
	2800	all	3a 2∗ 9r 5/4b	4848740
	3080	FRB,FBR,BFR,BRF	3a 1∗ 9r 4/3b	2625700
		RFB,RBF	3a 1∗ 8r 4/3b	2592588
	3360	all	3a 1∗ 8r 4/3b	2592588
	3500	all	3a 1∗ 8r 4/3b	2592588
DCT	1120	all	12a 8∗ 14r 10/10b	19958124
	1400	all	7a 6∗ 10r 8/8b	8838396
	1680	all	6a 4∗ 8r 6/6b	6998538
	1960	all	5a 4∗ 8r 6/6b	5731092

- TET is a constraint set by the user. (clock cycle = 280 ns)
- a: ALU, ∗: multiplier, r: register, b: input/output bus

As can be seen, the estimation order does affect the area prediction in some instances especially for the ARF and the EWF. These results were obtained without forcing any constraints on the resource count. The user can explore the design space further by constraining any class(es) of resources and invoking the estimator to predict the remaining resources and the overall layout area. Figure 7 depicts graphically the area estimates obtained for these three benchmarks shown in Table 3. For each benchmark, we compare three models for area estimation: Model (1) using FU+register area only, Model (2) using FU+register and a constant bus cost, and Model (3) using ChipEst. As noted before, ChipEst and CompEst have been benchmarked to estimate layout area and delay with about 10-15% accuracy [1], and therefore one can assume that they provide a reasonably close prediction of the physical design phase. It can be clearly seen that there are significant discrepancies between the three models. The sources of discrepancies between models (1) and (2), and model (3) can be attributed mainly to the following factors:

1. variation in component area and delay with aspect ratio: models (1) and (2) assume single values for area and delay per component (corresponding to unity aspect ratio) whereas ChipEst may select a different shape resulting in different area/delay not only for each component type, but also for each component *instance*.
2. physical design constraints: ChipEst produces different alternatives corresponding to different layout aspect ratios. This is illustrated by showing several area points for each design in the ChipEst model in Figure 7.
3. Wiring area: is not accounted for in (1), and only partially in (2).

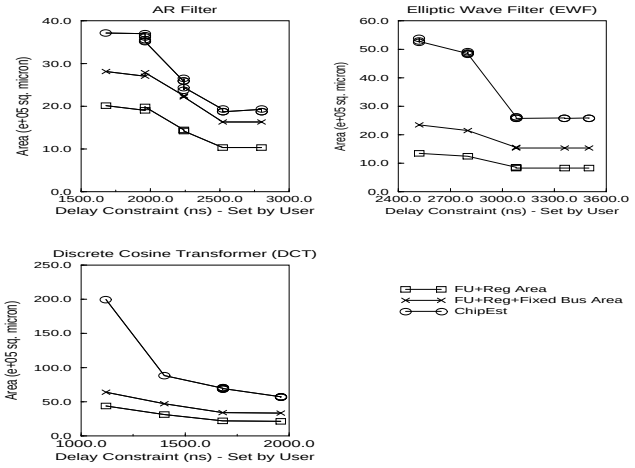


Figure 7: Area estimates for ARF, EWF, and DCT(multiple points indicate different aspect ratios).

4. Tiling effects: unused area which is not accounted for in (1) or (2).

These figures underline the importance of considering the layout effects on the overall area. Thus, it is clear that simple cost models do not perform well in assessing the overall design cost, and comprehensive models such as the proposed model are necessary to accurately explore the design space.

9 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 components, and to consider the variations in component area and delay for different physical implementations.

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.

Since no binding or scheduling are performed prior to or during estimation, there is no information on connectivity readily available to the estimator. Thus our system does not consider wiring 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] N. Dutt and C. Ramachandran, "Benchmarks for the 1992 High-Level Synthesis Workshop," *Tech. Report #92-107*, Information & Computer Science Department, UC Irvine, 1992.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] S. Chaudhuri and R. A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," *Proc. 7th International Workshop on High-Level Synthesis*, pp. 36-41, May 1994.

[9] 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.

[10] 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.

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

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

[13] S. Y. Ohm, F. J. Kurdahi, and N. Dutt, "Comprehensive Lower Bound Estimation from Behavioral Descriptions," *Proc. ICCAD '94*, pp. 182-186, Nov. 1994.

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

[15] 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.

[16] S. Chaudhuri, R. Walker, and J. Mitchell, "Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem," *IEEE Trans. VLSI Systems*, December 1994.

[17] P. Jha, C. Ramachandran, N. Dutt, and F. J. Kurdahi, "An Empirical Study on the Effects of Component Styles and Shapes on High-Level Synthesis," *Proc. VLSI'94*, 1994.

[18] J. P. Weng and A. C. Parker, "3D scheduling: High-Level synthesis with Floorplanning," *Proc. 5th International Workshop on High-Level Synthesis*, Buhlerhohe, Germany, 1991.

[19] S. Bakshi and D. Gajski, "A Strategy for Design Space Exploration," *Tech. Report #93-10*, Information & Computer Science Department, UC Irvine, 1993.

[20] S. Y. Ohm, F. J. Kurdahi, and N. Dutt, "A Unified Methodology for Estimating Resource Requirements for Early Design Space Exploration," *Tech. Report # 94-11-03*, ECE Department, UC Irvine, March 1994.

[21] P. J. Jha and N. Dutt, "The GENUS User Manual and C Programming Library," *Tech. Report #93-32*, Information & Computer Science Department, UC Irvine, 1994.