Comprehensive Lower Bound Estimation from Behavioral Descriptions *

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

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

Abstract

In this paper, we present a comprehensive technique for lower bound estimation (LBE) of resources from behavioral descriptions. Previous work has focused on LBE techniques that use very simple cost models which primarily focus on the functional unit resources. Our cost model accounts for storage resources in addition to functional resources. Our timing model uses a finer granularity that permits the modeling of functional unit, register and interconnect delays. We tested our LBE technique for both functional unit and storage requirements on several high-level synthesis benchmarks and observed near-optimal results.

1 Introduction

In order for high level design tasks such as High Level Synthesis (HLS) to produce reliable results, such tasks must rely on realistic and accurate models of hardware components. Without such realistic models, designs tasks essentially proceed in a blind fashion, which could result in designs not satisfying cost and/or timing constraints. Such an approach would result in unnecessary iterations through the design cycle and would increase the design turn around time and potentially, would decrease the competitiveness of the design method itself.

Much of the earlier design prediction work assumed the existence of *netlist-based* design descriptions as inputs, and hence produced netlist-based estimators [1]. While such tools provide an excellent feedback to the designer on the design quality, they can only be used *after* the design data path is synthesized, hence they provide *back-end* feedback. However, if 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 have studied the problem of providing *lower bound* estimates on resources given a control and data flow graph description of the design and a performance goal expressed as clock cycle constraint. The main features of our approach are the following:

1. It employs a comprehensive cost model which includes a realistic measure of both functional and storage 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 (FUs), such as adders. Table 1 shows such an example from the VTI 0.8μ cell library.
2. It is based on a unified paradigm which analyzes *lifetimes* of resources (be it FUs or registers) to determine lower bounds on these resources. This allows the user to analyze the tradeoffs of resource allocation.
3. It assumes a more accurate timing model which includes the delays of FUs, registers, and interconnects.

We have developed efficient algorithms and heuristics to support this model. Our initial experiments on standard HLS benchmarks [2] indicate that this model is quite accurate. Our model is more comprehensive than previous ones, and can be further expanded to account for additional physical design effects such as interconnect cost. Finally, this estimation scheme naturally lends itself to encapsulation within

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

Functional 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)

*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.

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.

2 Previous Work

There is some recent work for estimating lower bounds on area cost and total control steps (or csteps) [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. All of these works (with the exception of [7], [10], [11], and [12]) are mainly concerned with FUs in their area cost models. The work in [3] proposes a mathematical model for predicting the area-delay curve. [4] proposes an ILP formulation for lower bound estimation of performance given resource constraints. [5] addresses lower bounds on time and FU cost for functional pipelined data flow graph, but not register cost. [6] also addresses lower bounds on time and FU cost. It uses these two estimation algorithms to predict system level area-delay curve. However, it does not feature register cost estimation, either. An extension of the work in [3] is described in [7] and addresses lower bounds on time and area cost including interconnect cost, but not register cost. [8] presents a formal approach which seems to estimate FU cost better than [4] and [7] in some benchmarks. It considers the interdependency of the bounds of different FU types, but not registers in estimation. [9] finds the lower bound on FU cost and utilizes it in finding an optimal scheduling result effectively. [10] 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. [11] presents an integrated area-delay prediction model which includes FU, register, and interconect costs for use in system level partitioning. Finally, [12] 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 The Area Cost Estimation Algorithm

Figure 1 shows the overall structure of the area cost estimation algorithm **LBE**. 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 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 and register cost using

LBE()
{
parse input DFG;
read delay and area information;
read timing constraints including *clock period* and *maximum delay*;
total cstep number = $\lfloor$ (*maximum delay*) / (*clock period*) $\rfloor$;

for each operation O_i in input DFG,
determine $ASAP_i, ALAP_i, ASAP_i', ALAP_i'$;

Est_Area_Cost = Estimate_FU_Cost() + Estimate_Reg_Cost();

return(Est_Area_Cost);
}

Figure 1: Overall structure of the area cost estimation algorithm.

these time frames.

In this paper, we assume that the register cost is a *secondary cost* with respect to FU cost. That is, we assume that the FU cost estimation is performed before the register cost estimation as in the traditional methods, where scheduling is first performed to minimize the FU cost and subsequently values are assigned to registers so as to minimize register cost. In this sense, estimating the number of registers does not necessarily correspond to an absolute lower bound, but to a *conditional* lower bound on register cost subject to the estimated FU cost. Our scheme, however, is flexible to allow modification of the estimation order between FUs and registers.

Figure 2 shows an input DFG of the *differential equation* example which solves the 2nd order differential equation $y'' + 5xy' + 3y = 0$, and Figure 3 shows the initial time frames of the operations when the maximum delay is 80 ns and the clock period is 20 ns. In this example, we assume that the pre-defined transfer delay is 4.5 ns, and that additions, comparisons, and subtractions are executed by ALUs with 15 ns delay and multiplications by multipliers with 15 ns delay.

4 Estimating a Lower Bound on FU Cost

4.1 Basic idea

The basic idea behind the FU cost estimation scheme stems from the pigeon hole principle: if N operations are scheduled over K csteps, then it is guaranteed that at least $\lceil N/K \rceil$ operations are scheduled into some cstep among those K csteps. This can be stated slightly differently if we are talking about a cstep interval $Z = [X, Y]$ whose length is $Y - X + 1$. In this case, if N operations of type T are scheduled in the interval Z, then *at least* $\lceil N/(Y - X + 1) \rceil$ 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 $[ASAP_i, ALAP_i] \subseteq Z$. For each interval Z, we find the number of operations

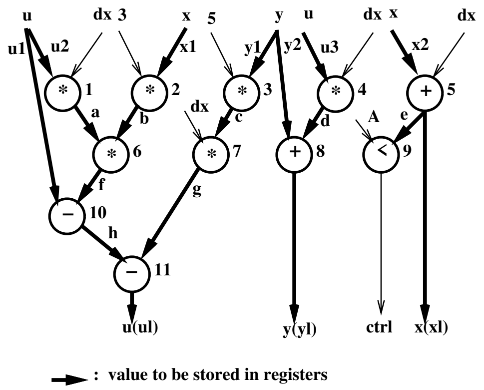

Figure 2: A DFG of the 2nd order differential equation example.

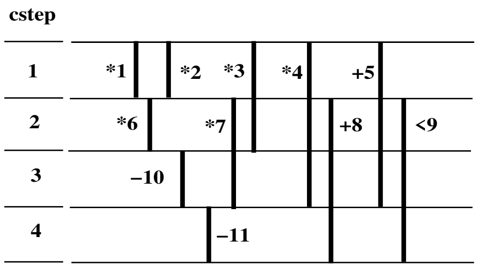

Figure 3: The initial time frames of the operations.

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.

For example, the four multiplications O_1, O_2, O_3, and O_6 in Figure 3 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 of the 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.

The above basic idea is generalized to support multicycling, chaining, and functional pipelining of operations. Figure 4 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 FU of type T.

As for the pipelined operations, we only have to count the first stage of them in estimating the lower bound on the number of the pipelined FUs [13]. Therefore, for each pipelined operation O_i, we assume that $ASAP'_i$ and $ALAP'_i$ are equal to $ASAP_i$ and $ALAP_i$ respectively, and then apply the same procedure above.

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

Estimate_FU_Cost()
{
    for each FU type T,
        $\psi(T)$ = set of operations of type T;
        for each cstep interval $Z \subseteq [1, total\ cstep\ number]$,
            for each operation O_i in $\psi(T)$,
                $P_{i,z} = [ASAP_i, ASAP'_i] \cap Z$;
                $Q_{i,z} = [ALAP_i, ALAP'_i] \cap Z$;
                $L_{i,z} = min(|P_{i,z}|, |Q_{i,z}|)$;
        $LB_T = max_z\ (\lceil (\sum_{O_i \in \psi(T)} L_{i,z})\ /\ |Z|\ \rceil)$;
    $Est\_FU\_Cost = \sum_T (LB_T \times Area_T)$;
    update Est_FU_Cost;
    return(Est_FU_Cost);
}

Figure 4: FU cost estimation algorithm.

4.2 Refining the lower bounds

Once the lower bounds on FU counts are estimated using the above procedure, we can use those bounds as an initial solution which is further refined to obtain tighter lower bounds. First, each FU is assumed to be used as many as the initial lower bound on the number of FUs of that particular type. This constraint may restrict the time frames of the operations to be scheduled.

For example, the initial time frame of O_3 is $[1, 2]$ as shown in Figure 3. 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. More sophisticated methods, which restrict the time frames effectively under the resource constraints, can be found in [13] and [14]. Figure 5 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 3.

If there is any adjustment in the time frames, we estimate the lower bounds again using the modified time frames to get tighter lower bounds. In this example, there is no update in the estimated FU cost because the initial lower bound estimation is exactly the same as the optimal one. This update procedure, however, is particularly effective in estimating a better register cost when it is a secondary cost with respect to FU cost as in the traditional design methods, since modifying the time frames of operations also affects the *lifetimes* of values between operations.

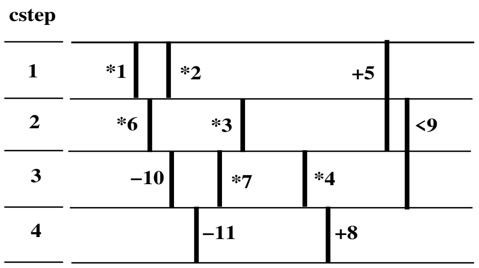

Figure 5: The adjusted time frames of the operations.

The time complexity in estimating the initial FU cost is $O(N \cdot C^2)$ and that in refining the estimation is roughly $O(N \cdot (N+E))$ [13], 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 $O(N \cdot (C^2 + N + E))$.

5 Register Cost Estimation

5.1 Basic technique

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

Figure 6 shows our register cost estimation algorithm. In this algorithm, the *weight* of a variable $V_{i,j}$, denoted by $W_{i,j}$, represents the *minimum* size of lifetimes 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. 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.

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

```
Estimate_Reg_Cost()
{
    for each variable V_{i,j}, calculate W_{i,j};
    Fanout_Reduction();
    Variable_Merging();
    for each interval Z ⊆ [1, total cstep number + 1],
        for each variable V_{i,j},
            if 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] ∩ Z;
            Q_{i,j,z} = [ALAP'_i + 1, ALAP'_i + W_{i,j}] ∩ Z;
            M_{i,j,z} = min(|P_{i,j,z}|, |Q_{i,j,z}|);
    LB_Reg = max_z (⌈ (Σ_{V_{i,j}} M_{i,j,z}) / |Z|) ⌉);
    Est_Reg_Cost = LB_Reg × Area_Reg;
    return(Est_Reg_Cost);
}
```

Figure 6: Register cost estimation algorithm.

for the variables. As mentioned before, the update procedure described in Section 4.2 helps increase their weights by restricting the time frames of operations, when the FU estimation is performed before register estimation. For further improvements, however, we apply two additional techniques: *Fanout Reduction* and *Variable Merging*. Table 2 shows the variables and their weights before and after these improvement techniques are applied.

Table 2: The weights of the variables.

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

5.2 Fanout reduction

Consider two shared variables $V_{i,j}$ and $V_{i,k}$, which have the same source O_i and thus represent the same data value. If O_j is guaranteed to be completed before O_k, then $V_{i,j}$ does not need to be considered when computing the register cost estimate, 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.

As an example, the two fanouts y_1 and y_2 in Figure 2 have the same source but different destinations: O_3 and O_8 respectively. For these shared variables, the condition "$ALAP'_3 \leq ASAP'_8$" is satisfied when the total number of csteps is 4 (see Figure 5). So we only have to consider the variable y_2 in estimating the register cost. In a similar way, u_1 and u_2 are reduced by u_3, x_1 by x_2, and e by xl respectively.

5.3 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. This modification helps increase $\sum_{V_{i,j}} M_{i,j,z}$ in Figure 6 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.

As an example, the two variables x_2 and xl in Figure 2 can be merged as a new variable, since the destination of x_2 is the source of xl. In this case, the initial weight of the new variable is 5, while the sum of the weights of the old two variables is 4, as shown in Table 2. As a result, our algorithm computes $M_{i,j,z}$ for the new merged variable as 1 when the interval $Z = [2, 2]$, while $M_{i,j,z}$s for the old variables are 0 respectively for the same interval. In a similar way, we can merge other variables as shown in Table 2.

The time complexity in calculating the weights of variables and in the improvement steps is roughly $O(E \cdot (N + E))$, and the complexity in estimating the register cost is $O(E \cdot C^2)$, 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 $O(E \cdot (C^2 + N + E))$.

6 Experimental Results

In order to validate the proposed lower bound estimation algorithms, we applied them to three well-known high level synthesis benchmarks from the HLS benchmark suite [2]: (1) the 2nd order differential equation, (2) the 5th order elliptic wave filter (EWF), and (3) the AR filter. Experimental results on these benchmarks are given in Tables 3, 4, 5, 6, and 7. In these experiments, we assumed that clock period is 20 ns and the total transfer delay including register and interconnect delay is 4.5 ns. The FU delays are specified in the tables. The CPU time for each experiment is less than 0.05 seconds on a SUN 4 workstation. Basically we compared our results with OASIC [10], ILP approach [15], HAL [16], and InSyn [17]. In many cases, however, we could find some better designs (denoted by 'Manual Design' in the tables) with less register cost than the designs reported in [15] and [16], though they are not optimal in some cases. As a result, these designs help us compare more accurately our estimates (especially register cost) with those of the actual (possibly optimal) designs obtained through the conventional scheduling and allocation processes.

Table 3: 2nd order differential equation

Delay (ns)	Actual Design			Our Estimation	
	Source	FU	Reg.	FU	Reg.
80 †	ILP [15]	2(+), 2(⋆)	4	2(+), 2(⋆)	4
80 ‡	Manual	2(+), 2(⋆)	5	2(+), 2(⋆)	5
120 ‡	Manual	2(+), 3(∗)	6	2(+), 3(∗)	6
140 ‡	Manual	2(+), 2(∗)	5	2(+), 2(∗)	5

†: The input variables are NOT stored in registers.
‡: The input/output variables are stored in registers.
⋆: multiplier (delay = 15.0 ns), ∗: multiplier (delay = 24.4 ns)
+: ALU (delay = 15.0 ns)

Table 4: 5th order elliptic wave filter - design I

Delay (ns)	HAL [16]		Manual Design		Our Estimation	
	FU	Reg.	FU	Reg.	FU	Reg.
340	3(+), 3(∗)	-	3(+), 3(∗)	11	3(+), 3(∗)	11
360	3(+), 2(∗)	-	2(+), 2(∗)	10	2(+), 2(∗)	10
380	2(+), 2(∗)	12	2(+), 2(∗)	9	2(+), 2(∗)	9
400	2(+), 2(∗)	-	2(+), 2(∗)	9	2(+), 2(∗)	9
420	2(+), 1(∗)	12	2(+), 1(∗)	9	2(+), 1(∗)	9

∗: multiplier (delay = 24.4 ns), +: adder (delay = 15.0 ns)

Table 5: 5th order elliptic wave filter - design II

Delay (ns)	OASIC [10]		HAL [16]		Our Estimation	
	FU	Reg.	FU	Reg.	FU	Reg.
340	3(+), 2($*_p$)	10	3(+), 2($*_p$)	12	3(+), 2($*_p$)	10
360	3(+), 1($*_p$)	10	3(+), 1($*_p$)	-	3(+), 1($*_p$)	10
380	2(+), 1($*_p$)	9	2(+), 1($*_p$)	12	2(+), 1($*_p$)	9

$*_p$: 2-stage pipelined multiplier (delay of 25.0 ns), +: adder (delay of 15.0 ns)

Table 6: 5th order elliptic wave filter - design III

Delay (ns)	InSyn [17]		Our Estimation	
	FU	Reg.	FU	Reg.
340	3(+), 2($*_p$)	8	3(+), 2($*_p$)	8
360	3(+), 1($*_p$)	-	3(+), 1($*_p$)	8
380	2(+), 1($*_p$)	8	2(+), 1($*_p$)	8
400	-	-	2(+), 1($*_p$)	8
420	-	-	2(+), 1($*_p$)	8
560	1(+), 1($*_p$)	9	1(+), 1($*_p$)	8

Note: The input variables are NOT stored in registers.
$*_p$: 2-stage pipelined multiplier (delay = 25.0 ns)
+: adder (delay = 15.0 ns)

Table 7: AR filter

Delay (ns)	ILP [15]		Manual Design		Our Estimation	
	FU	Reg.	FU	Reg.	FU	Reg.
120	-	-	4(+), 4(⋆)	6	4(+), 4(⋆)	6
140	-	-	4(+), 4(⋆)	6	4(+), 4(⋆)	4
160	-	-	2(+), 3(⋆)	6	2(+), 3(⋆)	4
180	2(+), 2(∗)	6	2(+), 2(⋆)	5	2(+), 2(⋆)	3

Note: The input variables are NOT stored in registers.
⋆: multiplier (delay = 15.0 ns), +: adder (delay = 7.5 ns)

The results are quite encouraging, indicating that the lower bound estimation algorithms achieve perfect accuracy with respect to estimating the functional unit requirements. The estimated lower bound on register count is also quite accurate as shown in the tables. We note here that the register cost estimate predicts a lower bound on register count *across all possible schedules that can be accomplished given the FU resources*. In most cases, the lower bound on register cost is equal to the actual register count. In other cases, the bound is 1 or 2 registers below the actual count. Note that no prior scheduling is assumed in our estimation.

Figure 7 shows the behavioral "shape function" that is generated for the EWF example (Table 4), and the predicted lower bound shape function. The area and delay figures for the adders, multipliers, and registers were generated using the VTI 0.8μ data path library described in Table 1. We note that the layout areas of a register and an adder are quite comparable in this library, indeed the register cost is slightly more than the adder's. This suggests that the register cost is as significant as the adder cost and must be considered in order to have realistic estimation of the overall design area.

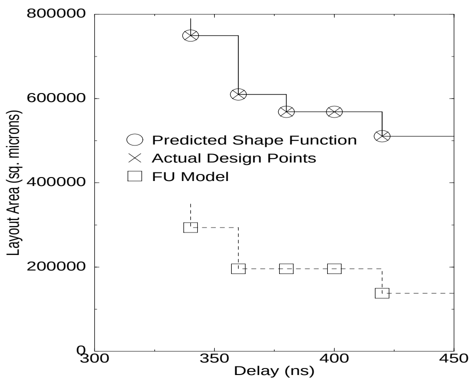

Figure 7: The actual and lower bound shape functions of the EWF with non-pipelined multiplier.

7 Conclusions

We presented an LBE technique that accounts for functional and storage units with a finer granularity of time, and presented experimental results of our approach on several HLS benchmarks. These results confirm the importance of accounting for both storage and functional units in lower bound estimation. Our estimates for functional unit and storage requirements are quite accurate and validate our approach for these examples.

As we move towards sub-micron technologies, the effects of interconnects will begin to dominate the design. Our present model does not estimate interconnect and multiplexing costs and delay, but the timing model can accommodate such estimates once available. Currently, the user provides some initial estimates of such delays. Once an RT level structure is further defined, it is possible to use accurate layout-based estimation schemes [1] to quickly get a better prediction of the interconnect delay before committing to a costly layout step. Clearly, better accounting for interconnect is needed and 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," Technical Report, ICS Department, UC Irvine, 1992.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[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. 91.

[12] 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.

[13] Seong Y. Ohm, Fadi J. Kurdahi, and Nikil Dutt, "A Unified Method for the Lower Bound Estimation on Resources," Technical Report, ECE Department, UC Irvine, 1994.

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

[15] M. Rim, R. Jain and R. D. Leone, "Optimal Allocation and Binding in High-Level Synthesis," *Proc. 29th DAC*, pp. 120-123, June 1992.

[16] P. G. Paulin and J. P. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis," *Proc. 26th DAC*, pp. 1-6, June 1989.

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