Lower-Bound Estimation for Multi-Bitwidth Time-Constrained Scheduling

Junjuan Xu[*+], Jason Cong[+], Xu Cheng[*]

[+]Computer Science Department, UCLA, Los Angeles CA 90095 USA

[*]Computer Science and Technology Department, Peking University, Beijing 100871 PRC

xujunjuan@mprc.pku.edu.cn, cong@cs.ucla.edu, chengxu@mprc.pku.edu.cn

***Abstract*--In high-level synthesis, accurate lower-bound estimation is helpful to explore the search space efficiently and to evaluate the quality of heuristic algorithms. For the lower-bound estimation of the scheduling problems, previous works mainly focus on the number of resources with uniform bitwidth. In this paper, we study the problem of lower-bound estimation on bitwidth summation of functional units for multi-bitwidth scheduling, where data-paths are composed of operations with various bitwidth. An integer linear programming (ILP) formulation and a polynomial time algorithm are presented. Experimental results indicate that the proposed algorithm produces good estimation, only 2% lower than the optimal results, which are obtained from ILP.**

I. INTRODUCTION

In high-level synthesis, scheduling is one of the central sub-tasks and has a pronounced impact on the area and performance of the final design. Given a data-flow graph (DFG), the scheduling task is to explicitly map operations onto control steps to achieve some goals while maintaining data-dependence and other constraints. Depending on the different constraints and optimization goals, the scheduling problems are divided into time-constrained scheduling (TCS), resource-constrained scheduling (RCS), and time- and resource-constrained scheduling (TRCS).

The decision problem of TRCS is well known to be NP-Complete [5]. Therefore, there is no existing polynomial algorithm for any of these scheduling problems. A wide variety of heuristic solutions are researched and developed to find good though not necessarily optimal schedules. To get a good knowledge of the quality of heuristics or help to explore the design space more efficiently, lower-bound estimation of resources, as well as performance, has been attracting interest from researchers. However, most of the related works only deal with those designs utilizing resources of uniform bitwidth.

Recently, there has been several works studying multi-bitwidth systems at the stage of high-level synthesis to get more chance of minimizing area of resources [2][3][4][7][8][9][10]. Since the area cost of resources is directly related to the bitwidth configuration, the objective becomes to minimize the total bitwidth for each resource. When the bitwidth of operations is various, and the scheduling begins to consider functional units having different bitwidth configurations, the problem is even harder. Most of these works proposed heuristics without any quantitative evaluation of optimality. To our knowledge, only the work in [4] presented an interval-based lower-bound estimation on the total bitwidth of functional units for TCS.

In this paper, we focus on the lower-bound estimation on bitwidth summation of functional units for the multi-bitwidth TCS problem. We transfer it to a minimizing memory job scheduling problem and solve it in polynomial time. An ILP formulation is presented to get the optimal solution and used to evaluate the accuracy of our proposed estimator. Also, LP relaxation is used to evaluate the accuracy in case of medium- and large-size benchmarks, where ILP cannot finish in acceptable time. Experimental results show that our estimator is close to the optimal results and more accurate than the interval-based algorithm in [4].

The paper is organized as follows. In Section II, we will introduce related preliminaries. A formal ILP formulation, as well as the estimator named *MB_UnitLength*, will be presented in Section III and IV. Section V shows the accuracy comparison, and Section VI concludes this paper.

II. PRELIMINARIES

Some concepts and definitions related to this problem are introduced first. The inputs of the multi-bitwidth TCS (MB-TCS) problem are a DFG and a time constraint. DFG is a directed acyclic graph, in which nodes represent operations, and edges represent data dependence. In the context of multi-bitwidth, each node is annotated with bitwidth information, which is set by users or obtained by bitwidth analysis tools, such as the technique proposed in [14]. Figure 1 shows an example of a multi-bitwidth DFG. The input-width constraint indicates a bitwidth requirement for a functional unit to perform the operation correctly.

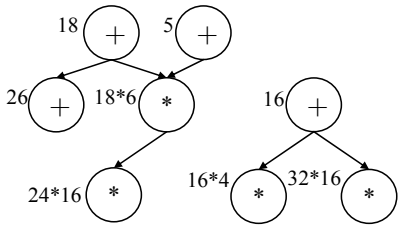

Figure 1. DFG with bitwidth information.

Only two types of operations, addition and multiplication, are considered in this work. We assume each operation o has two inputs with bitwidth pair (b_1, b_2), and define *the bitwidth of operation o* as $b(o) = max\{b_1, b_2\}$, for the same reason explained in [4]. For a group of operations O, we define *the bitwidth of set O* as the maximum bitwidth of all $o \in O$, denoted by $B(O) = max\{b(o) \mid o \in O\}$. Let $exe(o)$ denote the delay of an operation o, which is equal to the delay of the functional unit running o. In this work, we assume the execution time for adders and multipliers are 1 and 3 clock cycles, respectively. Let *the type of a functional unit* indicate its functionality, and F be the set of types that are available, addition and multiplication. *The type of an operation o*, $t(o)$, is determined by the type value $f \in F$, where o can be executed on functional units of type f. For a DFG under time constraint T, $ASAP(o)$ and $ALAP(o)$, the as-soon-as-possible and as-late-as-possible control steps of operation o are computed.

III. ILP FORMULATION

In this section, an ILP formulation is presented, which gives the tightest lower-bound on the summation of bitwidth for each type of functional units separately. ILP either maximizes or minimizes an objective function of a set of variables, subject to a group of linear equality and inequality constraints, and integral restriction on all of the variables.

Since the bitwidth of an instance of functional units of type f will be set as the maximum bitwidth of operations executed on it, we only need to consider those bitwidths associated with operations of type f, denoted by $B_f = \{b(o) \mid o \text{ is of type } f\}$.

Let $N_{f,b} \geq 0$ be the number of functional units configured with bitwidth b, where $b \in B_f$. The objective function for functional units of type f, which is the bitwidth summation of required functional units of type f, is defined as follows:

$$Min: \sum_{b \in B_f} b \cdot N_{f,b} .$$

Let variable $o_{c,b}$ represent whether operation o is scheduled at step c and bound to a functional unit with bitwidth b, where $c \in [ASAP(o), ALAP(o)]$, $b \in B_{t(o)}$ and $b \geq b(o)$. If this is the case, $o_{c,b}$ is set as 1; otherwise, the value is 0. Based on the definition, the expression representing whether o is scheduled at c is defined as follows:

$$r(o,c) = \sum_{b \in B_{t(o)} \,\&\, b \geq b(o)} o_{c,b} .$$

If the result of $r(o, c)$ is 1, o is scheduled at control step c.

The minimization is subject to four types of constraints. The first is the assignment constraint (A). Each operation must be scheduled at one and only one control step and executed on one and only one instance of functional units. The second is the resource constraint (R). The number of operations, which are bound to functional units with bitwidth b, is equal to or less than $N_{f,b}$ at any control step. The third is the data-dependence constraint (D), which means an operation cannot start execution until all of its predecessors have finished. The description of data-dependence is based on the structure proposed by Gebotys et al. in [6], which is proved tighter than the one used in [7]. The last constraint (I) is that all variables can only have non-negative integral values, and $o_{c,b}$ can only be set as either 1 or 0.

$$\sum_{\substack{c \geq ASAP(o) \\ c \leq ALAP(o)}} \sum_{b \in B_{t(o)} \,\&\, b \geq b(o)} o_{c,b} = 1 \qquad \forall o \in O \qquad \text{(A)}$$

$$\sum_{\substack{t(o)=f \\ ASAP(o) \leq c_1 \leq ALAP(o) \\ c_1 \leq c \leq c_1 + exe(o) - 1 \\ b(o) \leq b}} r(o, c_1) \leq N_{f,b} \qquad \forall c \in [1,T],\ \forall b \in B_f \qquad \text{(R)}$$

$$\sum_{\substack{c_2 \leq c + exe(o_1) - 1 \\ c_2 \geq ASAP(o_2) \\ c_2 \leq ALAP(o_2)}} r(o_2, c_2) + \sum_{\substack{c_1 \geq c \\ c_1 \geq ASAP(o_1) \\ c_1 \leq ALAP(o_1)}} r(o_1, c_1) \leq 1 \qquad \forall (o_1, o_2) \in E,$$

$$c \in [ASAP(o_1), ALAP(o_1)] \cap [ASAP(o_2), ALAP(o_2)] \qquad \text{(D)}$$

$$o_{c,b} \in \{0,1\}$$
$$N_{f,b} \in Z^{+} \qquad \forall o \in O, \forall c \in [1,T], \forall b \in B_f \qquad \text{(I)}$$

For a benchmark with 26 additions and 16 multiplications, when the time constraint is set as the length of the critical path, i.e., the minimum number of clock cycles a DFG needs, the formulation for adders comprises 551 variables and 203 equality and inequality constraints. As we relax the latency by 2 cycles, the number of variables and constraints increases to 993 and 335 respectively. The large number of variables and constraints causes ILP unsuitable for design space searching, where the estimation may be performed many times. Furthermore, it may not finish running in reasonable time for even one computation.

On the other side, this ILP formulation can produce a feasible schedule corresponding to the minimum requirement on the bitwidth summation of functional units of each type. Therefore, it achieves *the tightest lower-bound*. This is why we choose it as the baseline for the accuracy evaluation of our proposed estimator. Note that the lower-bounds for different types of functional units may not be achieved at the same time.

IV. MB_UNITLENGTH SOLUTION

A. Problem Transformation

We transform the original MB-TCS problem to a job scheduling problem by relaxing the data-dependence constraints and associating each job with a memory requirement, such that the lower-bound of the job problem is also that of the original DFG scheduling problem.

Different types of operations are processed separately. Let $O_f = \{o \mid t(o) = f\}$ be the set of operations of type f. In the new job scheduling problem, there is a job j corresponding to each $o \in O_f$. Let J represent the whole set of jobs. Note that operations of different types have unrelated job sets. A job j is associated with a triple, (*release*(j), *deadline*(j), *mem*(j)), representing its release time, deadline, and memory requirement on the processor, respectively. These values are set as *release*(j) = *ASAP*(o), *deadline*(j) = *ALAP*(o)+*exe*(o)-1, and *mem*(j) = b(o), where o is the corresponding operation of j. We define that *the memory configuration of a processor* is the maximum memory requirement of those jobs executed on it. The new problem, called **MIN_mem job scheduling problem**, is stated as:

Given: (1) A set of jobs with same execution time; (2) Each job has a release time and deadline; (3) Each job has a memory requirement for the allocated processor.

Objective: Allocate the jobs to a number of processors and schedule them under the constraints of release times and deadlines non-preemptively. The goal is to minimize the summation of the memory configuration of all the allocated processors.

In this way, data-dependence constraints are relaxed, while the integrity of the execution period is maintained. It is obvious to see that the solution space of the MIN_mem job scheduling problem is a superset of that of the original DFG scheduling, since the strict data-dependence is relaxed to release times and deadlines. Therefore, we have the following straightforward conclusion.

Theorem 1: The memory lower-bound of the MIN_mem job scheduling problem is also the bitwidth lower-bound of the corresponding functional units. □

B. Lower-Bound Calculation

In this sub-section, we will illustrate how to calculate the memory lower-bound for the MIN_mem job scheduling problem. For a job set S, we define *the maximum memory requirement of S* as $Mem(S) = max\{mem(j) \mid j \in S\}$, and *the minimum number of required processors for S under constraints of release times and deadlines* is denoted as $P\_Num(S)$.

We first divide J into sub-groups according to their memory requirement. All jobs in each sub-group have the same memory requirement. We then sort these sub-groups in the decreasing order of their memory requirement. Suppose the sorted sub-groups be $\{J_1, J_2, \ldots, J_k\}$, where k is the number of various memory requirement. From J_1, we know that the number of the processors with memory configuration $Mem(J_1)$ must be no less than $P\_Num(J_1)$ for any scheduling scheme of J, due to the definition of P_Num. Then for $J_1 \cup J_2$, the number of processors with memory configuration no less than $Mem(J_2)$ must be at least $P\_Num(J_1 \cup J_2)$. Combing these two facts, we conclude that except the $P\_Num(J_1)$ processors of memory configuration $Mem(J_1)$, the number of processors with memory no less than $Mem(J_2)$ must be at least $P\_Num(J_1 \cup J_2)$-$P\_Num(J_1)$. This calculation continues until J_k is included. Based on this observation, we come to the following conclusion.

Theorem 2: The *lower-bound of memory requirement* for the MIN_mem job scheduling problem is

$$\sum_{1 \le i \le k} \left[Mem\ (J_i) \cdot \left(P\_Num \left(\bigcup_{1 \le j \le i} J_j \right) - P\_Num \left(\bigcup_{1 \le j \le i-1} J_j \right) \right) \right],$$

where J_i, for $1 \le i \le k$, are the sorted sub-groups constructed in the aforementioned way, and we define that $P\_Num(\varnothing) = 0$. □

Then we will show how to compute the exact value of $P\_Num(S)$. The core function is the algorithm proposed by Simons in [13], for the following unit-time job scheduling problem.

Given: (1) A group of N unit-time jobs with fractional release times and deadlines; (2) m identical processors.

Objective: Determine whether a non-preemptive schedule exists under the constraints of release times and deadlines.

For additions with execution time of 1 clock cycle, Simons' algorithm can be directly used. For multiplications with execution time of 3 cycles, only trivial modification of the algorithm is required.

In the algorithm of Figure 2, a feasible upper-bound of P_Num is first calculated by using a simple earliest-release-first algorithm with time complexity $Nlog(N)$. Then a search is performed to obtain the minimum required processors. The searching method we used in this algorithm is to decrease the number of processors by one each time, until Simons' algorithm decides that no feasible schedule exists. The last feasible number is the exact value of P_Num. In experimental results, the times of Simons' algorithm being called vary between 1 and 4. The time complexity of Simons' algorithm is $O(mN^2)$. Since the jobs in our problem only have T distinct release times, the running time of Simons' scheduling is actually $O(Nlog(N)+mNT)$. Therefore, the total running time for the lower-bound calculation is $O(k(Nlog(N)+mNT))$ with P_Num being called k times.

```
Procedure P_Num
Input:
    N unit-time jobs with fractional release times and deadlines
Output:
    The minimum number of required processors

Calculate upper-bound of P_Num, denoted as ub;
ub --;
While Simons' scheduling algorithm returns feasible solution
        ub --;
ub++;
return ub;
```

Figure 2. Algorithm for the minimum number of processors.

Compared to the multi-bitwidth interval-based algorithm *MB_Interval* with time complexity $O(T^2(N+Nlog(N)))$ proposed in [4], we have the below result.

Theorem 3: For the operations of type f, O_f, the bitwidth lower-bound estimation of FUs given by *Theorem 1* and *Theorem 2* is more accurate than the multi-bitwidth interval-based algorithm, *MB_Interval*.

Proof: The proof consists of three steps.

Step 1: After relaxing the data-dependence of DFG to *ASAP* and *ALAP* steps without consideration of various bitwidths, the new problem is formulated as the below:

Given: (1) A group of operations A with same execution time; (2) Each operation has an integral release time and deadline;

Objective: Schedule these operations under the constraints of release times and deadlines such that the number of FUs is minimized.

For the above problem, the interval-based algorithm in [11], which is used in *MB_Interval*, gives a lower-bound estimation of the number of required FUs, while Simons' algorithm can give the optimal solution $P\_Num(A)$. We denote the result given by [11] as $Interval(A)$. Therefore, we have that $P\_Num(A) \geq Interval(A)$.

Step 2: In this step, we will give a formulation for the bitwidth lower-bound given by *MB_Interval*.

Divide operations O_f into sub-groups according to their bitwidth. All operations in each sub-group have the same bitwidth. Then sort these sub-groups in the decreasing order of their bitwidth. Suppose the sorted sub-groups be $\{A_1, A_2, \ldots, A_k\}$. From the algorithm of *MB_Interval*, it is easy to see that the number of FUs of bitwidth $B(A_1)$ is estimated as $Interval(A_1)$. We also have that the number of FUs, whose bitwidth is no less than $B(A_1 \cup A_2)$, is estimated as $Interval(A_1 \cup A_2)$. Therefore, the number of FUs of bitwidth $B(A_2)$ is $Interval(A_1 \cup A_2)$ - $Interval(A_1)$. In the same way, we can calculate the number of FUs of bitwidth $B(A_3),\ldots, B(A_k)$, respectively. Based on these observations, the lower-bound given by *MB_Interval* is denoted as:

$$LB_{INTERVAL} = \sum_{1 \le i \le k} \left[B(A_i) \cdot \left(Interval\left(\bigcup_{1 \le j \le i} A_j \right) - Interval\left(\bigcup_{1 \le j \le i-1} A_j \right) \right) \right] \quad (1)$$

where we define $N(\varnothing) = 0$.

Step 3: From *Theorem 1* and *Theorem 2*, the lower-bound given by *MB_UnitLength* is:

$$LB_{UNIT} = \sum_{1 \le i \le k} \left[Mem(J_i) \cdot \left(P\_Num\left(\bigcup_{1 \le j \le i} J_j \right) - P\_Num\left(\bigcup_{1 \le j \le i-1} J_j \right) \right) \right] \quad (2)$$

Also, we define $P\_Num(\varnothing) = 0$.

Formulation (1) and (2) can be transformed into:

$$LB_{INTERVAL} = \sum_{1 \le i \le k-1} \left[Interval\left(\bigcup_{1 \le j \le i} A_j \right) \cdot (B(A_i) - B(A_{i+1})) \right] + Interval\left(\bigcup_{1 \le j \le k} A_j \right) \cdot B(A_k) \quad (1')$$

$$LB_{UNIT} = \sum_{1 \le i \le k-1} \left[P\_Num\left(\bigcup_{1 \le j \le i} J_j \right) \cdot (Mem(J_i) - Mem(J_{i+1})) \right] + P\_Num\left(\bigcup_{1 \le j \le k} J_j \right) \cdot Mem(J_k) \quad (2')$$

From the definition of J and Mem, we know that J_i corresponds to A_i, and $B(A_i) = Mem(J_i)$, $1 \le i \le k$. Also, since $\{A_1, A_2, \ldots, A_k\}$ is constructed in decreasing order of bitwidth, $B(A_i)$ is greater than $B(A_{i+1})$, $1 \le i \le k-1$. From formulation (1') and (2'), we can come to the conclusion that $LB_{INTERVAl} \le LB_{UNIT}$. □

V. EXPERIMENTAL RESULTS

The experiments are conducted in a C++/UNIX environment. A LP/ILP solver from [1] is used to obtain the results of ILP and LP relaxation. We will use ILP formulations, which are shown to produce the tightest lower-bounds in Section III, as the baseline to evaluate the accuracy of the estimator, *MB_UnitLength*. For medium- and large-size benchmarks, the large number of variables and constraints causes the solver unable to find the solution in ten hours. Therefore, we choose nine small benchmarks to perform the comparison with ILP, and nine real-life benchmarks to perform the comparison with LP relaxation.

For the unit-time job scheduling problem, Simons proposed two algorithms in [12] and [13] with time complexity of $O(N^3 loglog(N))$ and $O(mN^2)$, respectively. The time complexity of these two algorithms for our problem in this paper is actually $O(TN^2 loglog(N))$ and $O(Nlog(N)+mNT)$ due to the same reason as explained in Section IV(B). Because the algorithm of [12] is easier to implement, we used it in our experiments.

To understand the impact of the data-dependence constraints on the lower-bound accuracy, we also perform experiments on another form of ILP, which omits the inequality constraints corresponding to data-dependence, denoted by *ARI*. Actually, *ARI* partially relaxes the data-dependence constraints to *ASAP* and *ALAP* steps. The full ILP formulation is denoted by *ARDI*, and the interval- based algorithm in [4] is denoted as *MB_Interval*.

Table 1. Accuracy comparison with ARDI.

	ARDI	ARI	MB-UnitLength	MB-Interval[4]
adder	1	1	1	1
multiplier	1	1	98%	95%
adder (s)	3.04	1.47	0.001	0.001
multiplier (s)	24.92	15.40	0.001	0.001

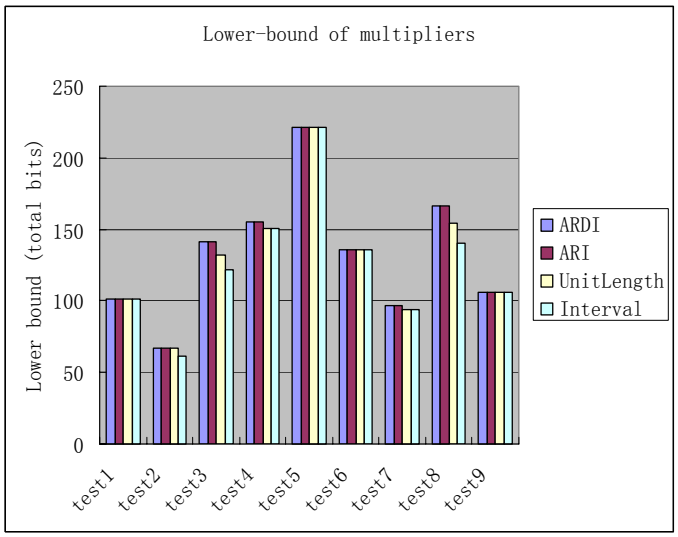

Figure 3. Comparison of lower-bounds for multipliers (Small benchmarks).

In Table 1, the second and third rows of list the accuracy comparison of *ARDI*, *ARI*, *MB_UnitLength* and *MB_Interval*, averaging over nine small benchmarks. The last two rows list the average running time for adders and multipliers. We can see that *ARI* is as tight as *ARDI*. For adders, all of the four estimators give the same results, meaning that *ARI*, *MB_UnitLength* and *MB_Interval* achieve the tightest lower-bounds for all these benchmarks. For multipliers, *MB_UnitLength* is only 2% lower than *ARDI* on average, higher than *MB_Interval*. Figure 3 shows the detailed results of multipliers for each benchmark.

The LP relaxation of *ARDI* is denoted by *ARD*, and the one omitting the data-dependence is *AR*. For those benchmarks that *ARDI* can be solved in acceptable time, the result of *ARD* is 2% lower than *ARDI* on average. This means that the accuracy of *ARD* is high enough to be used as the baseline to evaluate the efficiency of the lower-bound estimations.

Table 2. Accuracy comparison with ARD.

	ARD	AR	MB-UnitLength	MB_Interval [4]
adder	1	99.7%	1	1
multiplier	1	99.6%	99.1%	90.2%
add-time(s)	53.88	2.86	0.35	0.04
mul-time(s)	24.88	7.24	0.12	0.07

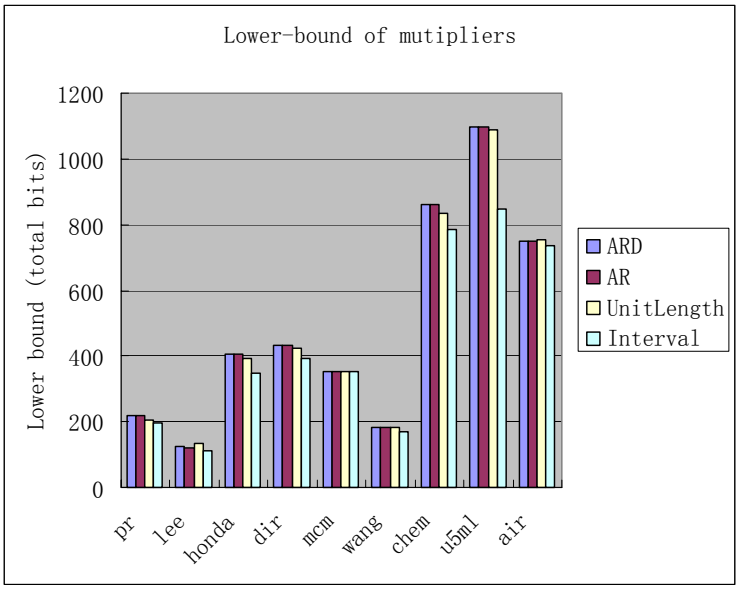

Figure 4. **Comparison of lower-bounds for multipliers (Real-life benchmarks).**

Table 2 shows the average comparison results and running time. *AR* is still very close to *ARD*, only 0.4% difference observed with much less running time. For multipliers, *MB_UnitLength* is up to 30% higher than *MB_Interval* and is 9% better on average. For adders, *MB_Interval* is much faster than *MB_UnitLength*. This is because the algorithm of [12] requires long runtime for large benchmarks. If the algorithm of [13] is used, the runtime of *MB_UnitLength* should be better. Figure 4 shows the detailed results of multipliers for each benchmark.

Based on the above experimental results, we come to the following conclusions. Lower-bounds by relaxation of data-dependence to *ASAP* and *ALAP* steps are very close to optimal results. For functional units with execution time of multiple clocks, *MB_Interval* gives the loosest lower-bound. The main reason is that besides data-dependence relaxation, *MB_Interval* does a further estimation on the minimum number of required resources, *P_Num*, while *MB_ UnitLength* does not. For functional units with execution time of one single clock, both of the two estimators are very accurate.

VI. CONCLUSION

Traditional lower-bound estimations at the scheduling stage of high-level synthesis only consider functional units with uniform bitwidth. In this work, we studied the problem of lower-bound estimation on total bitwidth of functional units for the multi-bitwidth TCS. In particular, we have presented a formal ILP formulation and a polynomial time heuristic algorithm. For lower-bound estimation, experimental comparisons indicate that our algorithm produces accurate estimations.

ACKNOWLEDGEMENTS

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

REFERENCE

[1] ftp://ftp.ics.ele.tue.nl/pub/lp_solve.

[2] V. Agrawal, A. Pande and M. M. Mehendale, "High Level Synthesis of Multi-Precision Data Flow Graphs," *Proc. of the 14th International Conference on VLSI Design*, 2001.

[3] G. A. Constantinides, P. Y. K. Cheung and W. Luk, "Heuristic Datapath Allocation for Multiple Wordlength Systems," *Proc. of Design*, *Automation and Test in Europe*, 2001.

[4] J. Cong et al, "Bitwidth-Aware Scheduling and Binding in High-Level Synthesis," *Proc. of ASPDAC*, 2005.

[5] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *Freeman*, 1979.

[6] C. H. Gebotys and M. I. Elmasry, "Optimal VLSI Architectural Synthesis," *Kluwer Academic Publishers*, 1992.

[7] C. T. Hwang, J. H. Lee and Y. C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis," *IEEE Trans. On Computer Aided Design*, 1991.

[8] K. Kum and W. Sung, "Combined Word-Length Optimization and High-Level Synthesis of Digital Signal Processing Systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2001.

[9] S. Mahlke et al, "Bitwidth Cognizant Architecture Synthesis of Custom Hardware Accelerators," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2001.

[10] M. C. Molina, J. M. Mendias and R. Hermida, "High-Level Synthesis of Multiple-Precision Circuits Independent of Data-Objects Length," *Proc. of the 39th Design Automation Conference*, 2002.

[11] A. Sharma and R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applications," *IEEE Trans. on VLSI Systems*, 1993.

[12] B. Simons, "Multiprocessor Scheduling of Unit-time Jobs with Arbitrary Release Times and Deadlines," *SIAM J. Computers*, 1983.

[13] B. Simons and M. Warmuth, "A Fast Algorithm for Multiprocessor Scheduling of Unit-Length Jobs," *SIAM J. Computers*, 1989.

[14] M. Stephenson, J. Babb and S. Amarasinghe, "Bitwidth Analysis with Application to Silicon Compilation," *Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2000.

[15] J. Xu, J. Cong and X. Cheng, "Lower-Bound Estimation for Multi-Bitwidth Scheduling," *Proc. of ISCAS*, 2005.