Lower Bound Estimation of Hardware Resources for Scheduling in High-Level Synthesis

Shen Zhaoxuan and Jong Ching Chuen

School of Electrical and Electronic Engineering
Nanyang Technological University, Nanyang Avenue, Singapore 639798

E-mail: eccjong@ntu.edu.sg

Received January 28, 2002; revised April 1, 2002.

Abstract In high-level synthesis of VLSI circuits, good lower bound prediction can efficiently narrow down the large space of possible designs. Previous approaches predict the lower bound by relaxing or even ignoring the precedence constraints of the data flow graph (DFG), and result in inaccuracy of the lower bound. The loop folding and conditional branch were also not considered. In this paper, a new stepwise refinement algorithm is proposed, which takes consideration of precedence constraints of the DFG to estimate the lower bound of hardware resources under time constraints. Processing techniques to handle multi-cycle, chaining, pipelining, as well as loop folding and mutual exclusion among conditional branches are also incorporated in the algorithm. Experimental results show that the algorithm can produce a very tight and close to optimal lower bound in reasonable computation time.

Keywords lower bound estimation, chaining, pipelining, mutual exclusion, high-level synthesis

1 Introduction

High-level synthesis is the realization of register transfer level (RTL) structure from the system functional specification. It consists of two major tasks, scheduling and allocation, the former determines the assignment of operations to control steps while the latter binds operations to hardware resources. The main difficulty in high-level synthesis is how to select the best design from the large number of possible designs. Trade-offs on design space exploration and optimization goals become the crucial problem in high-level synthesis.

In synthesis of large systems, lower bound prediction not only can narrow down the design space by pruning lots of inferior designs but also enables the synthesis system to explore the large design space efficiently[1−3]. It is extremely time consuming for the synthesis system to locate directly at several "good" points in the design space without actually synthesizing all the possible designs. Furthermore, the accurate lower bound estimation results can be used to evaluate the quality of designs synthesized by heuristic algorithms.

There were some previous studies on lower bound predictions before actual scheduling. Jain *et al.*[4] formulated a mathematical model for area-delay prediction for high-level synthesis. Timmer *et al.*[5] generated the area-delay curves by relaxing some of the precedence constraints to select the minimal cost module set. Nourani and Papachristou[6] gave a layout area cost estimation algorithm for a given RTL datapath with standard-cell and full-custom layout methodologies. Execution interval analysis approach is widely used for scheduling and estimation. Timmer and Jess[7] adopted the bipartite graph matching algorithm for resource constraints scheduling and estimation while Sharma and Jain[8] estimated architecture resource and performance by calculating the minimal overlap among different execution intervals of operations. Wehn *et al.*[9] obtained the hardware lower bounds with a simple mobility matrix calculation. Ohm *et al.*[10] proposed a fanout reduction and a variable merging approach to refine the lower bound on registers and an integrated approach for lower bound estimation with an area cost model covering register, buses as well as functional units[11]. Hu *et al.*[12] extended their interval estimation method to functional pipeline. Another popular model in high-level synthesis is the integer linear programming (ILP) formulation[13−16]. Rim and Jain[17,18]] derived the lower bound using a relaxed ILP formulation and a greedy algorithm. Chaudhuri and Walker[19] gave another

ILP formulation to compute the lower bound on functional units by considering the interdependence of the bound on different functional unit (FU) types while Langevin and Cerny[20] tried to improve Rim and Jain's relaxation algorithm by adopting a recursive greedy technique to compute the ASAPUC (as soon as possible under constraint) value. Shen and Jong[21] proposed an integer programming model with a surrogate relaxation technique for the lower bound and upper bound when scheduling and multicomponent selection were considered simultaneously. Other models and techniques such as parameterized component estimation[22], timing and re-timing analysis and estimation[23,24] and power consumption estimation[25] have been proposed to aid the high-level synthesis.

The quality of a lower bound prediction depends on the accuracy of the lower bound and the efficiency of the estimator. The previous approaches estimated the lower bound by relaxing or even ignoring the precedence constraints of the Data Flow Graph (DFG), thus resulted in inaccuracy of the lower bound. Neither the loop folding nor conditional branch was considered. In this paper, a stepwise refinement algorithm is proposed to predict the lower bound on the number of hardware resources under the time constraints, taking the precedence constraints of the DFG into account. Experimental results show that the proposed algorithm can provide a very tight lower bound in a reasonable time. It can also handle multi-cycle, chaining, pipelining, as well as loop folding. A new condition tree naming and matching algorithm is also proposed to handle the mutual exclusion among conditional branches.

The rest of the paper is organized as follows. Section 2 describes the estimation algorithm. Section 3 discusses the variations on the active range graph due to the DFG precedence constraints and presents the stepwise refinement estimation algorithm. Section 4 extends the estimation algorithm to handle chaining, pipelining and loop folding. A new condition tree naming and matching algorithm for estimating the architectural resources with mutual exclusion among conditional branches is presented in Section 5. Experimental results are shown in Section 6 and Section 7 concludes the paper.

2 Estimating the Lower Bound on the AR Graph

2.1 Definitions

The following definitions are used to describe the estimation and stepwise refinement algorithms.

Definition 1 (ASAP and ALAP). *ASAP time of an operation node in the DFG is the earliest time step in which the node can be executed subject to the precedence constraints of the DFG and the amount of the available resources.*

ALAP time of an operation node is the latest time step in which the operation must be completed so that all the operations in the DFG can be finished by a given time constraint T.

Let the starting time step of the DFG be 0. The initial ASAP and ALAP time can be obtained under the assumption, where the amount of available resource is unlimited.

Definition 2 (Active range (AR)). *The active range of an operation node is the time interval between its ASAP time and ALAP time, denoted by [ASAP, ALAP].*

Definition 3 (Active range graph (AR graph)). *The active range graph of an operation type is the graph that depicts all the active ranges of this type of operations in the DFG.*

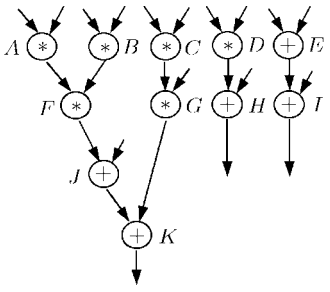

Fig.1. Differential equation DFG.

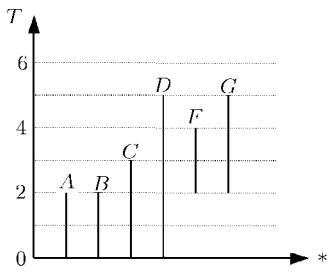

Fig.2. AR graph of multiplication.

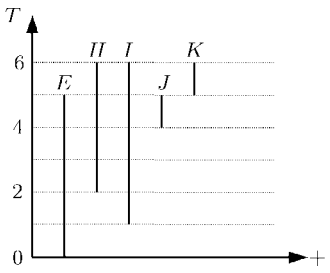

Fig.3. AR graph of addition.

Fig.1 shows the DFG of the differential equation example from [26]. The active range graphics of

the multiplication operations and the addition operations are shown in Fig.2 and Fig.3 respectively. Here it is assumed that the addition operation takes one cycle and the multiplication operation takes two while the time constraint is 6.

The following notations are adopted:

If $A = \{a_1, a_2, \ldots, a_i, \ldots, a_n\}$ refers to a set, $|A|$ refers to the number of elements in the set and $a_i (1 \leq i \leq |A|)$ refers to one of its elements.

O_{op} refers to the set of all the operation nodes of the type op in DFG, and TC_{op} is the cycle time of the type op.

ASAP(a_i) and ALAP(a_i) denote the ASAP and ALAP time of the operation node a_i.

$[a, b]$ is a time interval and $A \cap [a, b]$ is the set of all the nodes $a_i \in A$ which satisfy $a \leq \text{ASAP}(a_i) \leq \text{ALAP}(a_i) \leq b$.

2.2 Lower Bound on the AR Graph

The active range graph partly reflexes the precedence relations among the operations in the DFG. It is used to estimate the lower bound of the number of resources for each type of operation. The algorithm is described as follows.

Algorithm 1. Estimating Lower Bound of Resources under the Time Constraint T

```
For each operation in the DFG Do
{
    Calculate its ASAP & ALAP under the assumption that there are unlimited resources and the
    ending time of the DFG is T
}
For each operation type op Do
{
    Sort ASAP time of all the operation nodes of type op in ascending order and put into S
    Sort ALAP time of all the operation nodes of type op in ascending order and put into L
    LB_op = 0
    For i = 1 To |S| Do
    For j = 1 To |L| Do
    {
    If (l_j − s_i ≥ TC_op) Then //TC_op is the execution steps of operation type op
    {
    K = O_op ∩ [s_i, l_j]    // Find all operations of type op in [s_i, l_j]
    m = ⌈|K|/(l_j − s_i)⌉    // Compute the minimum no. of operations in each step of [s_i, l_j]
    If (m > LB_op) Then LB_op = m
    }
    }
}
```

For multi-cycle operations, $m = \lceil |K| / (\lfloor (l_j - s_i)/TC_{\text{op}} \rfloor) \rceil$ instead of $m = \lceil |K|/(l_j - s_i) \rceil$.

Table 1 and Table 2 show the processing steps of estimating the lower bound of the resources for multiplications and additions in the DFG of the differential equation example under the time constraint of 6. The worst complexity of the algorithm is $O(n^2)$ where n is the number of nodes in the DFG.

Table 1. Estimation of Multiplication

$[s_i, l_j]$	$T_v = l_j - s_i$	Operations $*$	$\lvert K\rvert$ = # of op	$T_s = \lfloor Tv/TC \rfloor$	$m = \lceil \lvert K\rvert / Ts \rceil$
$[0,2]$	2	A, B	2	$\lfloor 2/2 \rfloor = 1$	$\lceil 2/1 \rceil = 2$
$[0,3]$	3	A, B, C	3	$\lfloor 3/2 \rfloor = 1$	$\lceil 3/1 \rceil = 3$
$[0,4]$	4	A, B, C, F	4	$\lfloor 4/2 \rfloor = 2$	$\lceil 4/2 \rceil = 2$
$[0,5]$	5	A, B, C, D, F, G	6	$\lfloor 5/2 \rfloor = 2$	$\lceil 6/2 \rceil = 3$
$[2,4]$	2	F	1	$\lfloor 2/2 \rfloor = 1$	$\lceil 1/1 \rceil = 1$
$[2,5]$	3	F, G	2	$\lfloor 3/2 \rfloor = 1$	$\lceil 2/1 \rceil = 2$
$LB_* =$	MAX $(m) =$				3

Table 2. Estimation of Addition

$[s_i, l_j]$	$T_v = l_j - s_i$	Operations +	$\lvert K\rvert$ = # of op	$T_s = \lfloor Tv/TC \rfloor$	$m = \lceil \lvert K\rvert /Ts \rceil$
$[0,5]$	5	E, J	2	$\lfloor 5/1 \rfloor = 5$	$\lceil 2/5 \rceil = 1$
$[0,6]$	6	E, H, I, J, K	5	$\lfloor 6/1 \rfloor = 6$	$\lceil 5/6 \rceil = 1$
$[1,5]$	4	J	1	$\lfloor 4/1 \rfloor = 4$	$\lceil 1/4 \rceil = 1$
$[1,6]$	5	H, I, J, K	4	$\lfloor 5/1 \rfloor = 5$	$\lceil 4/5 \rceil = 1$
$[2,5]$	3	J	1	$\lfloor 3/1 \rfloor = 3$	$\lceil 1/3 \rceil = 1$
$[2,6]$	4	H, J, K	3	$\lfloor 4/1 \rfloor = 4$	$\lceil 3/4 \rceil = 1$
$[4,5]$	1	J	1	$\lfloor 1/1 \rfloor = 1$	$\lceil 1/1 \rceil = 1$
$[4,6]$	2	J, K	2	$\lfloor 2/1 \rfloor = 2$	$\lceil 2/2 \rceil = 1$
$[5,6]$	1	K	1	$\lfloor 1/1 \rfloor = 1$	$\lceil 1/1 \rceil = 1$
$LB_+ =$	MAX $(m) =$				1

3 Stepwise Refinement Estimation Algorithm

3.1 Variations on the AR Graph

Taking a further consideration of the precedence constraints in the DFG, it is noticed that the estimation Algorithm 1 does not give accurate results. This is because the ASAP time and ALAP time are calculated under the assumption, where the hardware resources are unlimited. If the estimated number of resources is used to re-calculate the ASAP and ALAP time, the active ranges of some operations are changed. This will lead to a new estimation and will give a tighter lower bound. The new lower bound will in turn lead to another iteration of AR graph variation and re-estimation. After several iterations, the active range graph of the DFG will no longer change and a very accurate lower bound that satisfies the precedence constraints in the DFG will be obtained.

There are three variations on the AR graph: the range overriding variation, the path overriding variation and the successive paths variation.

(1) Range overriding variation

Having estimated that at least 3 multipliers are needed as shown in Table 1, it can be seen from Fig.2 that the operations A, B and C must be executed between $[0,2]$ and the operation D cannot be executed in $[0,2]$. Hence, the active range of D must be cut off $[0,2]$ and changed to $[2,5]$. This is called range overriding variation or range overriding cutting.

(2) Path overriding variation

Consider the example of the DFG in Fig.4, with the AR graph shown in Fig.5 and a time constraint of 5, if it is estimated that only one adder is needed, the earliest time step for operation D is 4 because it takes at least 4 steps to complete all of its predecessors. So the active range of D must be cut off the frame $[3,4]$ and changed to $[4,5]$. This is the path overriding variation or path overriding cutting.

(3) Successive paths variation

After the range overriding variation and path overriding variation, all the active ranges of the operations in the successive paths need to be modified accordingly. For the differential equation example, as the active range of operation D has been cut off the frame $[0,2]$ and changed to $[2,5]$ due to the range overriding, the active range of successive operation H must be cut off the frame $[2,4]$ and changed to $[4,6]$ accordingly. This is the successive paths variation or successive paths cutting.

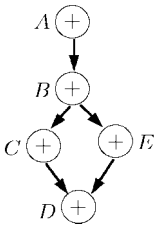

Fig.4. A DFG example.

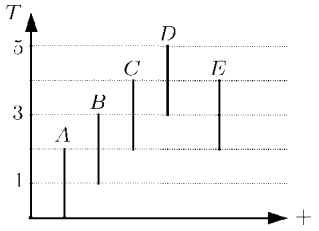

Fig.5. AR graph of Fig.4.

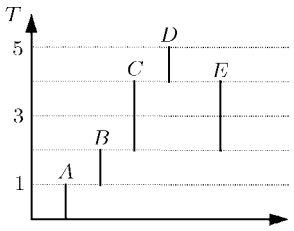

Fig.6. AR graph after variation.

Corresponding to the ASAP and ALAP calculation, each variation has two kinds of cutting: the lower frame cutting (due to the ASAP time overriding) and the upper frame cutting (due to the ALAP time overriding). The estimated number of resources is used to perform the above three variations

on the ASAP and the ALAP time. In the example of Fig.5, with the estimated one adder for the DFG, it needs to modify the ASAP time of D to 4 and cut off the lower frame $[3,4]$ from its active range due to the path overriding. Similarly, due to the path overriding, the ALAP time of B must be changed to 2 and its upper frame $[2,3]$ must be cut off from its active range because it takes at least 3 steps to finish all of its descendants. The upper frame cutting of B will in turn result in the upper frame cutting of operation A due to the successive paths propagation and the ALAP time of A must be changed to 1. The last feasible AR graph for the DFG of Fig.4 is depicted in Fig.6. The stepwise refinement estimation algorithm based on the variations on the AR graph is given in the following section.

3.2 Stepwise Refinement Estimation Algorithm on the Lower Bound of Resources

Algorithm 2. Stepwise Refinement Lower Bound Estimation under Time Constraint T

```
Sort operation types by their costs in descending order and put into the set OP
For i=1 To |OP| Do
{
    Call Algorithm 1 to estimate the number of resource LB_i for op_i
    Repeat
    Save the current AR graph of op_i
    While there is ASAP range or path overriding under current {LB_j} (1 ≤ j ≤ i) Do
    {
        Do the lower frame cutting
        Propagate the variation to the successive paths
    }
    While there is ALAP range or path overriding under current {LB_j} (1 ≤ j ≤ i) Do
    {
        Do the upper frame cutting
        Propagate the variation to the successive paths
    }
    For j = 1 To i Do
    {
        Call Algorithm 1 to re-estimate the number of resource NewLB_j for op_j
    }
    changed = False
    If (NewLB_i > LB_i) Then {LB_i++; changed = True; }
    Else {
              For j = 1 To i Do
              {
              If (NewLB_j > LB_j) Then {LB_i++; changed = True; Break;}
              }
         }
        If (changed) Then Restore the saved old AR graph of op_i
    Until (Not changed)
}
```

4 Lower Bound on Chaining, Pipelining and Loop Folding

4.1 Lower Bound with Chaining Operations

The algorithm can be extended to process chaining operations. In operation chaining, more than one operations are scheduled one control step, i.e., the operations are chained. Fig.7 shows a DFG where a multiplication $(*)$ requires 80ns and an addition $(+)$ takes 35ns. The cycle time is assumed to be 80ns. By chaining two additions into one cycle as shown in Fig.8, the DFG can be scheduled in one C-step, which is faster than the regular schedule shown in Fig.7, where two C-steps are required.

The estimation algorithm for chaining operations only requires a small modification in Algorithm A: change the ASAP, ALAP and TC_{op} from the unit of C-step to the unit of actual clock period.

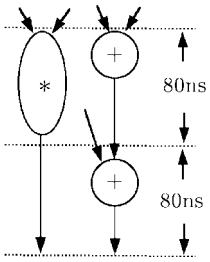

Fig.7. Regular schedule.

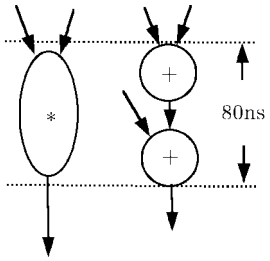

Fig.8. Chaining.

4.2 Lower Bound with Pipelined Resources

In contrast to operation chaining, operations can be pipelined and scheduled to more than one control steps. Pipelined operations have to be executed by pipelined function units, which execute operations in multi-stages. The problem of scheduling operations to pipelined resources is based on the fact for which once a stage of a pipelined functional unit is empty, it is available for other operations. Fig.9 is a schedule of the Differential Equation with a 2-stage pipelined multiplier. In the example, two independent multiplications can be scheduled into two consecutive C-steps (e.g., D and A) because they are executed at the different stages of the pipelined multiplier.

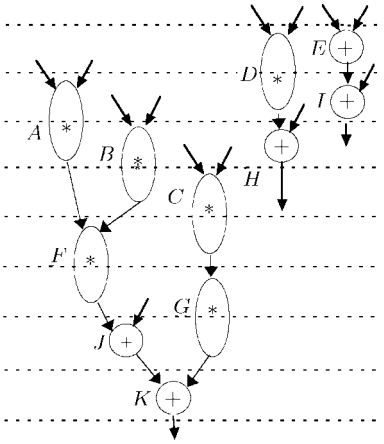

Fig.9. Schedule of differential equation with pipelined multiplier.

To obtain the lower bound with pipelined resources, first it needs to estimate the number of pipelined resources that can be pipelined for the full stage of the time interval. Then the remaining operations are used to estimate the number of non-pipelined resources. The algorithm is described as follows.

Algorithm 3. Estimating Lower Bound with Pipelined Resources

For each operation in the DFG Do
{
  Calculate its ASAP and ALAP under the assumption that there are unlimited resources and the ending time of the DFG is T
}
For each operation type op Do
{
  Sort ASAP time of all the operation nodes of type op in ascending order and put into S
  Sort ALAP time of all the operation nodes of type op in ascending order and put into L
    $LBP_{\text{op}} = LBNP_{\text{op}} = 0$
    For $i = 1$ To $|S|$ Do
    For $j = 1$ To $|L|$ Do
    {
    If $(l_j - s_i \geq TC_{\text{op}})$ Then
    {
      $K = O_{\text{op}} \cap [s_i, l_j]$ // Find all operations of type op in $[s_i, l_j]$
      $x = MAX\_MACHINE\_NUMBER$
      For $b = s_i$ To l_j Do // Find minimum pipelined resources needed in $[s_i, l_j]$
        $x = \text{Min}\,(|K \cap [b, b + TC_{\text{op}}]|, x)$
      $stage = l_j - s_i - 1$;
      If $(stage + 1 > TC_{\text{op}})$ Then $y = \lfloor |K|/stage \rfloor$
      Else $y = 0$ // Calculate resources needed if all operations are pipelined
      $m_{\text{pipe}} = \text{Min}\,(x, y)$ // Obtain the pipelined resources needed
      $leftop = |K| - m_{\text{pipe}} * stage$ // Calculate the number of operations left
      $m_{\text{nonpipe}} = \lceil leftop/(\lfloor (l_j - s_i)/TC_{\text{op}} \rfloor) \rceil$ // Get the number of non-pipelined resources

```
            If (Cost(m_pipe, m_nonpipe) > Cost(LBP_op, LBNP_op)) Then
            {// Accept the estimated resource number with the minimum cost
              LBP_op = m_pipe
              LBNP_op = m_nonpipe
            }
          }
      }
  }
```

4.3 Lower Bound with Loop Folding

Apart from processing chaining operations and pipelined resources, the algorithm can also be extended to estimate the lower bound of resources with loop folding. Loop folding[27−30] is widely used to exploit the concurrency beyond the iteration boundaries. It adopts the pipeline technique to parallel different iterations of the loop.

The following definitions are used:

Iteration Time (IT) is the number of C-steps, which are needed for the completion of an iteration.

Initiation Interval (II) is the number of C-steps between the start of two successive pipelined iterations.

Loop Carried Dependency (LCD) is the precedence dependency among operations across different iterations.

The estimation algorithm for loop folding is as follows: first if there exists a loop carried dependency (LCD), the minimal latency (i.e., initiation interval II) of the pipeline is determined. Then multiple active ranges that can be executed simultaneously in the pipeline are used to estimate the lower bound of the resources. The algorithm is described as follows.

Algorithm 4. Estimating Lower Bound for Loop Folding under Iteration Time Constraint IT

```
For each operation in the DFG Do
{
    Calculate its ASAP and ALAP under the assumption whene there are unlimited resources and the
    ending time of the DFG is T
}
II = 0
For each pair of operation nodes n_i, n_j which have LCD between them Do
    II = Max(II, abs(asap(n_i) − asap(n_j)) + 1) // Find the minimum iteration interval
For each operation type op Do
{
    Sort ASAP time of all the operation nodes of type op in ascending order and put into S
    Sort ALAP time of all the operation nodes of type op in ascending order and put into L
    LB_op = 0
    For i = 1 To |S| Do
        For j = 1 To |L| Do
        {
        If (l_j − s_i ≥ TC_op) Then
        {
            K = ∅
            For k = 0 To ⌊IT/II⌋ Do
            { // Calculate all the intervals that can execute at the same time due to loop folding
                s = mod(s_i + k * II, IT)
                l = mod(l_j + k * II, IT)
                K = K ∪ (O_op ∩ [s_i, l_j])
            }
            m = ⌈|K|/(⌊(l_j − s_i)/TC_op⌋)⌉ // Calculate the minimum resources needed
            If (m > LB_op) Then LB_op = m
        }
    }
}
```

5 Condition Tree Naming and Matching for Estimation with Mutual Exclusion

Wakabayashi and Tanaka[31] proposed a one-hot encoded vector to process the mutual exclusion between different conditional branches. A hot-bit is assigned to every conditional branch and the hot-bit of un-overlaped conditions cannot be re-used. This will lead to a very exhaustive vector if there exist a large number of conditional statements.

Here a new condition tree naming and matching approach is proposed to process the mutual exclusion between different conditional branches. First a unique identification is assigned to each condition node and each condition branch is named with a branch name, which consists of the unique identification of the condition node followed by a condition status bit. Then each operation node is named with a condition path name, which is made up of a series of branch names indicating the path traversing along the condition tree from the root to the node. Each operation node name can have different length and an operation node name matching algorithm is developed to check whether two operation nodes are mutually exclusive. A clique partitioning algorithm is adopted to detect the groups of nodes which are mutually exclusive among each other in the group. Each of these groups is viewed as a single operation node during the estimation. The method is described in the following sections.

5.1 Condition Tree Naming

(1) Condition node naming

Assign a unique identification (UI) to each condition node. F_i is assigned to a fork node and J_i is assigned to a join node.

(2) Condition branch naming

Each branch following a condition fork node is named as UI-SB, i.e., the UI of the fork node followed by a condition status bit (SB).

For 'If $\sim$ Then $\sim$ Else' condition, SB = 0 refers to the false branch and SB = 1 refers to the true branch.

For 'Case' condition, SB = 0 $\sim$ 9 refers to each branch of the 'Case' condition. If there are more than 10 branches, multiple nested 'Case' conditions can be used.

(3) Operation node naming

Each operation node is named with a branch name series $\{\mathrm{UI}-\mathrm{SB}\}^*$, indicating a path from the root condition node to the node.

If an operation node is not within any conditional branch, it is named as X.

(4) Simplifying operation node name

Form the end to the beginning of the operation node name, if a J_i is found, then the string between F_i to J_i (including F_i and J_i) is eliminated from the operation node name.

5.2 Mutual Exclusion Detection

(1) Checking mutual exclusion between two operation nodes

Let ↑NameP be the pointer to a name string.

GetNextUIOf(Name, ↑NameP) returns a UI in the Name to which ↑NameP currently points and then moves ↑NameP to the position behind the UI.

GetNextSBOf(Name, ↑NameP) returns an SB in the Name to which ↑NameP currently points and then moves ↑NameP to the position behind the SB.

The mutual exclusion detection algorithm is shown as follows.

```
Stop = False
While (Not Stop) Do
{
    Str1 = GetNextUIOf(Name1, ↑NameP1)
    Str2 = GetNextUIOf(Name2, ↑NameP2)
    If (Str1 == NULL or Str2 == NULL) Then Stop = True
```

If ($Str1 \neq Str2$) Then Return No
$Str1 =$ GetNextSBOf(Name1, ↑NameP1)
$Str2 =$ GetNextSBOf(Name2, ↑NameP2)
If ($Str1$ == NULL or $Str2$ == NULL) Then $Stop =$ True
If ($Str1 \neq Str2$) Then Return Yes
}
Return No

(2) Detecting the maximum mutual exclusion operation node group

An undirected graph is introduced in which the node refers to the operation node and the connection between two nodes indicates the two operation nodes are mutually exclusive. We adopt the clique partitioning algorithm described in [32] to detect the maximum mutual exclusion operation node group.

5.3 Estimating Lower Bound with Mutual Exclusion

Algorithm 5. Estimating Lower Bound with Mutual Exclusion under Time Constraint T

For each condition node in the DFG Do
{
assign a unique name
assign a branch name to each of its branch
}
For each operation node in the DFG Do
{
get its node name and make correspondent simplification
}
For each operation node in the DFG Do
{
Calculate its ASAP and ALAP under the assumption where there are unlimited resources and the ending
time of the DFG is T
}
For each operation type op Do
{
Sort ASAP time of all the operation nodes of type op in ascending order and put into S
Sort ALAP time of all the operation nodes of type op in ascending order and put into L
$LB_{\text{op}} = 0$
For $i = 1$ To $|S|$ Do
For $j = 1$ To $|L|$ Do
{
If ($l_j - s_i \geq TC_{\text{op}}$) Then
{
$K = O_{\text{op}} \cap [s_i, l_j]$
// Find maximum mutual exclusive operation node groups
$F =$ number of maximum mutual exclusive operation node groups in K
// As all the mutual exclusive operations can share the resources, these operations
// can be treated as only one operation. It needs to deduct them from the number of
// operations in the interval:
$m = \lceil (|K| - |F|)/(\lfloor (l_j - s_i)/TC_{\text{op}} \rfloor) \rceil$
If ($m > LB_{\text{op}}$) Then $LB_{\text{op}} = m$
}
}
}

6 Experiments and Results

The lower bound estimation algorithms were implemented on a SUN workstation in C language.

Several benchmarks were tested and the results were compared to some previously published results.

6.1 Comparison of the Lower Bounds with Previous Results

The first example depicted in Table 3 is the benchmark of AR lattice filter taken from [4]. It has 16 multiplications and 12 additions. It was assumed that each multiplication needs two C-steps while each addition needs one. It can be seen from the table that at $T = 12 \sim 13$, the lower bound $(4*, 2+)$ obtained is tighter than Jain's $(3*, 1+)$ at $T = 12$[4] and Sharma and Rim's $(3*, 2+)$ at $T = 13$[8,18]. The lower bound $(3*, 1+)$ at $T = 15 \sim 17$ was also tighter than $(2*, 1+)$ at $T = 16$ produced by Jain.

Table 3. Comparison of Lower Bounds on AR Filter

	Jain's estimation				Sharma and Rim's estimation					Our estimation				
T	11	12	16	32	11	13	14	18	34	11–13	14	15–17	18–33	34
$*$	4	3	2	1	4	3	3	2	1	4	3	3	2	1
$+$	2	1	1	1	2	2	1	1	1	2	1	1	1	1

The second example shown in Table 4 is the benchmark of fifth-order elliptical wave (EW) filter, which contains 26 additions and 8 multiplications. It was also assumed that a multiplication needs two C-steps while an addition needs one. The table shows that a better result of $(1*, 2+)$ at $T = 21 \sim 27$ was obtained as compared to the results from Jain[4] and Sharma[8] and Rim[18], who had $(1*, 1+)$ at $T = 26$ and $T = 27$. In addition, at $T = 19 \sim 20$, $(2*, 2+)$ was obtained while the results were not available in Jain, Sharma and Rim's reports.

Table 4. Comparison of Lower Bounds on EW Filter

	Jain's estimation		Sharma and Rim's estimation				Our estimation			
T	17	26	17	18	21	27	17	18–20	21–27	28
$*$	3	1	3	2	1	1	3	2	1	1
$+$	3	1	3	2	2	1	3	2	2	1

6.2 Comparison of the Estimation with Actual Schedule Results

Table 5 shows the lower bound on the differential equation, which has been discussed in the previous sections. The estimation results obtained are compared with those of Jain, Sharma and Rim and also the actual schedule obtained by a Precedence-Bipartite scheduler developed. It can be seen that all the lower bounds obtained for this example are the optimal results because they are the same as the actual schedule generated by our scheduler. It is also noted that the lower bound $(4*, 1+)$ at $T = 6$ obtained by Jain, Sharma and Rim is not practical because they use a more costly multiplier instead of an adder.

Table 5. Comparison of Estimation with Actual Schedule on Differential Equation

	Jain's estimation		Sharma and Rim's estimation			Our estimation				Our actual schedule			
T	6	12	6	7	13	6	7	8–12	13	6	7	8	13
$*$	4	1	4	2	1	3	2	2	1	3	2	2	1
$+$	1	1	1	2	1	2	2	1	1	2	2	1	1

Table 6 compares the lower bounds for the example of EW filter with the actual schedule obtained by our scheduler and Paulin's HAL system[26]. A multiplier (delay = 80ns) needs one cycle (delay = 80ns) while two additions (delay = 35ns) were chained into one cycle. All the lower bounds obtained are the same as the actual schedule.

Table 6. Comparison of Estimation with Schedule on EW Filter with Operation Chaining

Delay	HAL's schedule[26]		Our schedule		Our estimation	
680ns	3*	3+	3*	3+	3*	3+
720ns	2*	3+	2*	2+	2*	2+
760ns	2*	2+	2*	2+	2*	2+
840ns	1*	2+	1*	2+	1*	2+
1120ns	NA		1*	1+	1*	1+
*: Multiplier (delay = 80ns), +: Adder (delay = 35ns), Cycle (delay = 80ns)						

In fact, the lower bounds obtained here for both the EW filter and the differential equation are the optimal ones[33].

6.3 Lower Bound with Pipelined Resources

Table 7 shows the lower bound of the differential equation with pipelined resources. It can be seen that the lower bound obtained is better at $T = 6$ because a non-pipelined multiplier is used instead of a more expensive pipelined multiplier. Besides, a lower bound at $T = 5$ was obtained while there was no result reported by Jain, Sharma and Rim.

Table 7. Lower Bounds on Differential Equation with Pipelined Multipliers

	Jain's estimation	Sharma and Rim's estimation		Our estimation		
T	6	6	8	5	6	8
$*p$	2	2	1	2	1	1
$*$	0	0	0	2	1	0
$+$	1	1	1	1	1	1

6.4 Lower Bound on Loop Folding

Table 8 is the benchmark of 16-point FIR filter[34], which has no Loop Carried Dependency (LCD). Several Initial Interval (II) values were used. All the lower bounds obtained under the different Iteration Time (IT) and Initial Interval (II) are the same as the actual schedules generated by the "Theda.fold" system[27] and are proved to be optimal[28].

Table 8. Lower Bounds of 16-Point FIR Filter with Loop Folding

IT	II	Our estimation		Theda. Fold's schedule[27]	
6	1	15+	8*	15+	8*
6	2	8+	4*	8+	4*
6	3	5+	3*	5+	3*
6	4	4+	2*	4+	2*
7	5	3+	2*	3+	2*
10	8	2+	1*	2+	1*
16	15	1+	1*	1+	1*

6.5 Lower Bound on Conditional Branch

For the conditional branch, the conditional tree naming and matching algorithm presented in Section 5 is to find the mutual exclusion between conditional branches. Table 9 shows the lower bounds on some famous benchmarks that contain conditional branches. From the table, it can be seen that the lower bounds obtained are the same as the actual schedule generated and they are the optimal.

Table 9. Lower Bounds on Conditional Branch

Examples	T	Our estimation		Our schedule		Difference
MAHA[35]	6	1+	1−	1+	1−	0
Sehwa[34]	5	1+	2−	1+	2−	0
	6	1+	1−	1+	1−	0
Kim[36]	7	2+	1−	2+	1−	0
	8	1+	1−	1+	1−	0

Our CPU time for obtaining the lower bound is also very short and ranged from 10–50ms, compared to that of Jain, Rim and Sharma, for 20ms, 40ms and 500ms, respectively.

In summary, the lower bounds, which we produce for different benchmarks, are either as good as or better than those produced by the previous estimating systems. They are also close to the actual schedules and hence close to optimal ones.

7 Conclusion

As the lower bound estimation of hardware resources is useful and important to high-level synthesis systems and it allows the synthesis system to explore a large design space without having to implement the final designs, it is crucial that the estimation gives tight and accurate results. In this paper, a

new stepwise refinement method is proposed. It takes into consideration of precedence constraints of the DFG to predict the lower bound on the number of resources under time constraints. The algorithm includes the capability to handle multi-cycle, chaining, pipelining, loop folding and the mutual exclusion of conditional branches. Experimental results obtained on testing several benchmarks show that the lower bounds produced are very tight and close to optimal while the CPU time required for the estimation is very short.

References

[1] Hu Y, Carlson B S. A unified algorithm for the estimation and scheduling of data flow graphs. *Journal of Circuits Systems and Computers*, June, 1996, 6(3): 287–318.

[2] Tiruvuri G, Chung M. Estimation of lower bounds in scheduling algorithms for high-level synthesis. *ACM Trans. Design Automation of Electronic Systems*, Apr., 1998, 3(2): 162–180.

[3] Narasimhan M, Ramannjam J. On lower bounds for scheduling problems in high-level synthesis. In *Proc. 2000 Design Automation Conference*, Los Angeles, CA, USA, June, 2000, pp.546–551.

[4] Jain R, Parker A C, Park N. Predicting system-level area and delay for pipelined and non-pipelined designs. *IEEE Trans. CAD-ICAS*, 1992, 11(8): 955–965.

[5] Timmer A H, Heijligers M J M, Jess J A G. Fast system-level area-delay curve prediction. In *Proc. 1st Asian Pacific Conference on Hardware Description Languages, Standards and Applications*, Brisbane, Australia, Dec., 1993, pp.198–207.

[6] Nourani M, Papachristou C. A layout estimation algorithm for RTL datapaths. In *Proc. 30th Design Automation Conference*, Dallas, TX, USA, 1993, pp.285–291.

[7] Timmer A H, Jess J A G. Execution interval analysis under resource constraints. In *Proc. International Conference on Computer Aided Design*, Santa Clara, CA, USA, Nov., 1993, pp.454–459.

[8] Sharma A, Jain R. Estimating architectural resources and performance for high-level synthesis applications. In *Proc. 30th Design Automation Conference*, Dallas, TX, USA, 1993, pp.355–360.

[9] Wehn N, Glesner M, Vielhauer C. Estimating lower bounds in high-level synthesis. In *IFIP Trans. VLSI 93*: *Proc. IFIP TC10/WG10.5 Int. Conf. VLSI*, Grenoble, France, Sept., 1993, pp.261–270.

[10] Ohm S Y, Kurdahi F J, Dutt N. Comprehensive lower bound estimation from behavioral descriptions. In *Proc. International Conference on Computer Aided Design*, San Jose, CA, USA, Nov., 1994, pp.182–187.

[11] Ohm S Y, Kurdahi F J, Dutt N D. A unified lower bound estimation technique for high-level synthesis. *IEEE Trans. CAD-ICAS*, May, 1997, 16(5): 458–472.

[12] Hu Y, Ghouse A, Carlson B S. Lower bounds on the iteration time and the number of resources for functional pipelined data flow graphs. In *Proc. International Conference on Computer Design*, Cambridge, MA, USA, Oct., 1993, pp.21–24.

[13] Hwang C T, Lee J H, Hsu Y C. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. CAD-ICAS*, 1991, 10(4): 464–475.

[14] Gebotys C H, Elmasry M I. Global optimization approach for architectural synthesis. *IEEE Trans. CAD-ICAS*, Sept., 1993, 12(9): 1266–1278.

[15] Hwang C T, Hsu Y C. Zone scheduling. *IEEE Trans. CAD-ICAS*, Jul., 1993, 12(7): 926–934.

[16] Wilson T C, Grewal G W, Banerji D K. An ILP solution for simultaneous scheduling, allocation, and binding in multiple block synthesis. In *Proc. International Conference on Computer Design*, Cambridge, MA, USA, 1994, pp.581–586.

[17] Rim M, Jain R. Estimating lower-bound performance of schedules using a relaxation technique. In *Proc. International Conference on Computer Design*, Cambridge, MA, USA, 1992, pp.290–294.

[18] Rim M, Jain R. Lower-bound performance estimation for the high-level synthesis scheduling problem. *IEEE Trans. CAD-ICAS*, Apr., 1994, 13(4): 451–458.

[19] Chaudhuri S, Walker R A. Computing lower bounds on functional units before scheduling. In *Proc. 7th International Symposium on High-Level Synthesis*, Ontario, Canada, May, 1994, pp.36–41.

[20] Langevin M, Cerny E. A recursive technique for computing lower-bound performance of schedules. In *Proc. International Conference on Computer Design*, Cambridge, MA, USA, 1993, pp.16–20.

[21] Shen Z X, Jong C C. Functional area lower bound and upper bound on multicomponent selection for interval scheduling. *IEEE Trans. CAD-ICAS*, July, 2000, 19(7): 745–759.

[22] Jha P K, Dutt N D. Rapid estimation for parameterized components in high-level synthesis. *IEEE Trans. VLSI systems*, Sept., 1993, 1(3): 296–303.

[23] Mintz D, Dangelo C. Timing estimation for behavioral descriptions. In *Proc. 7th International Symposium on High-Level Synthesis*, Ontario, Canada, May, 1994, pp.42–47.

[24] Rabaey J M, Potkonjak M. Estimating implementation bounds for real time DSP application specific circuits. *IEEE Trans. CAD-ICAS*, June, 1994, 13(6): 669–683.

[25] Kruse L, Schmidt E, Jochens G, Stammermann A, Nebel W. Lower bound estimation for low power high-level synthesis. In *Proc. 13th Int. Symposium on System Synthesis*, Madrid, Spain, 2000, pp.180–185.

[26] Paulin P G, Knight J P. Force-directed scheduling for the behavioral synthesis of ASIC's. *IEEE Trans. CAD-ICAS*, June, 1989, 8(6): 661–679.

[27] Lee T F, Wu A C H, Lin Y L, Gajski D D. A transformation-based method for loop folding. *IEEE Trans. CAD-ICAS*, Apr., 1994, 13(4): 439–450.
[28] Lee T F, Wu A C H, Gajski D D, Lin Y L. An effective methodology for functional pipelining. In *Proc. International Conference on Computer Aided Design*, Santa Clara, CA, USA, 1992, pp.230–233.
[29] Hwang C T, Hsu Y C, Lin Y L. PLS: A scheduler for pipeline synthesis. *IEEE Trans. CAD-ICAS*, Sept., 1993, 12(9): 1279–1286.
[30] Passos N L, Sha E H M, Bass S C. Loop pipelining for scheduling multi-dimensional systems via rotation. In *Proc. 31st Design Automation Conference*, San Diego, CA, USA, 1994, pp.485–490.
[31] Wakabayashi K, Tanaka H. Global scheduling independent of control dependencies based on condition vectors. In *Proc. 29th Design Automation Conference*, Anaheim, CA, USA, 1992, pp.112–115.
[32] Springer D L, Thomas D E. Exploiting the special structure of conflict and compatibility graphs in high-level synthesis. *IEEE Trans. CAD-ICAS*, July, 1994, 13(7): 843–856.
[33] Timmer A H, Heijligers J M, Stok L, Jess J A G. Module selection and scheduling using unrestricted libraries. In *Proc. of the European Conference on Design Automation with the European Event in ASIC Design*, Paris, France, Feb., 1993, pp.547–551.
[34] Park N, Parker A C. Sehwa, a software package for synthesis of pipelines from behavioral specifications. *IEEE Trans. CAD-ICAS*, Mar., 1988, 7(3): 356–370.
[35] Parker A C, Pizarro J T, Milnar M. MAHA: A program for datapath synthesis. In *Proc. 23rd Design Automation Conference*, Las Vegas, NV, USA, 1986, pp.461–466.
[36] Kim T K, Yonezawa N, Liu W S J, Liu C L. A scheduling algorithm for conditional resource sharing — A hierarchical reduction approach. *IEEE Trans. CAD-ICAS*, Apr., 1994, 13(4): 425–438.

Shen Zhaoxuan received the B.Sc. degree in computer science from Zhejiang University, China and the M.Eng. degree in electrical and electronic engineering from Nanyang Technological University, Singapore. From August 1987 to January 1994, he was a research associate at Laboratory of CAD & Graphics, Institute of Computing Technologies, Chinese Academy of Sciences, Beijing, China. During 1996–1999, he was a senior engineer at Institute of High Performance Computing Singapore, developing parallel optimization algorithm for VLSI floorplanning, placement and synthesis, etc. He was a senior software engineer at Arcadia Design Systems, Inc. San Jose, CA, USA from March. 1999 to November 2000 and a senior engineer at Synopsys Inc. Mountain View, CA, USA from December 2000 to January 2002. He is now with Cadence Design Systems Inc. San Jose, CA, USA, developing new generation of P&R tools for SOC VLSI design. Dr. Shen received the 1991 First Class Science & Technology Progressing Award from Chinese Academy of Sciences for the contribution to the EDCADS tool development. He has been an IEEE member since 1996.

Jong Ching Chuen received the BSc (Eng) degree in electronics with computer science and the PhD degree in electronic engineering from Queen Mary College, University of London, U.K., in 1983 and 1988 respectively. From July 1987 to October 1990, he worked in the area of high-level synthesis of digital systems first in University of Southampton, U.K. and then in Racal Research Limited, U.K. In 1991, he joined the Nanyang Technological University, Singapore, as a faculty member. He is now an associate professor in the Division of Circuits and Systems, School of Electrical and Electronic Engineering. Dr. Jong is a chartered engineer, a member of IEE and a member of BCS. His technical interests include high-level synthesis, ASIC design and fast-prototyping of digital designs.