

Fast System–Level Area–Delay Curve Prediction

Adwin H. Timmer, Marc J.M. Heijligers and Jochen A.G. Jess

Eindhoven University of Technology, Department of Electrical Engineering
Design Automation Section, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

In this paper a unified approach of lower bound functional area and cycle budget estimations is presented to predict area–delay characteristics of designs at system level. The estimations are mainly based on relaxing precedence constraints in a behavioral design description and are the most accurate estimations reported to date.

1. Introduction

In high–level synthesis, a data path consisting of modules (i.e. functional units), registers and interconnection units is synthesized from a behavioral design description [McFa90]. Such a description is represented by a data flow graph (DFG), which is a translation of an algorithmic specification in a hardware description language [Eijnd92]. The ability to predict the area–delay characteristics of designs without actually implementing them is important in producing quality designs in a reasonable time, and is therefore an important part of an (interactive) system design environment [Fleu93]. If a design will be part of a larger system, then it is important to select one or more 'good' points out of the design space without synthesizing all possibilities. So an accurate lower bound area–delay curve can speed up the design space search and can furthermore be used to evaluate the quality of a design.

Previous approaches ([Chen91], [Jain92], [Shar93], [Timm93a]) determine a lower bound estimation of the module set with minimal area for each possible number of cycle steps. For a full area–delay curve the number of these estimations and hence the CPU time can become rather large. For a full design space exploration it should furthermore be possible to select freely from *unrestricted* libraries containing module types with a large variety in delay, area and so on for each operation type (instead of an one–to–one mapping of operation types to module types). The NEAT (New Eindhoven Architectural synthesis Toolbox) system is capable of performing area estimations using such libraries [Timm93a]. However, unrestricted libraries with their range of delays for each operation type also increase the design space and the total number of area estimations.

The area estimations will often lead to the same module set for different time constraints. Based on this observation we present a unified approach of lower bound area and cycle budget estimations to predict the area–delay characteristics of a design (see figure 1). The approach starts with the derivation of the smallest module set possible and the lower bound on the number of cycles for that module set. [Rim92], [Shar93] and [Timm93b] give approaches to estimate a lower bound on the cycle budget for a given module set. The next step consists of decrementing the number of cycles and determining the corresponding lower bound module set. For this new module set the lower bound on the number of cycles is derived. The process is repeated until the smallest number of cycles possible has been reached. In this approach (which is implemented in the NEAT system) each module set is derived only once which leads to a run time efficient prediction of the area–delay curve.

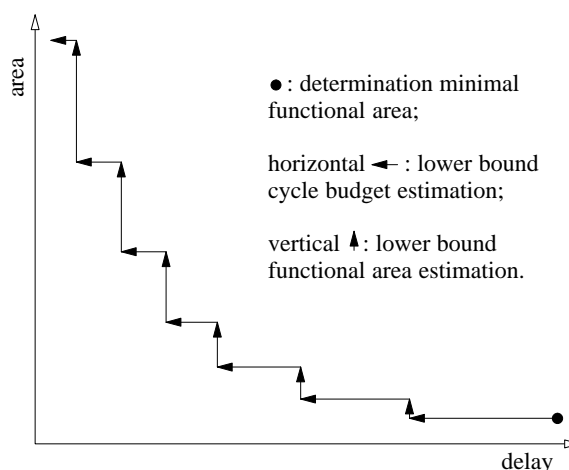


Fig 1: Fast Area–Delay Curve Prediction.

The cycle budget estimation is also a partial check for the feasibility of the derived module sets. If the lower bound estimation of the cycle budget exceeds the number of cycles for which the module set was derived, then the module set cannot be correct. In that case the module set can be reviewed until the lower bound estimation of the cycle budget does not exceed the time constraint anymore. This additional and novel analysis can improve the accuracy of the functional area estimation considerably.

The remaining paper starts with some definitions in section 2. The lower bound module selection and the lower bound cycle budget estimation approaches of the NEAT system are presented in section 3 and section 4. In section 5 adaptations are given in case a selected module set turns out to be infeasible. In section 6 experiments and results are presented.

2. Definitions

A DFG is a tuple (V, E) , where V is the set of nodes (operations) and E the set of edges representing dependencies between operations. T is the set of available cycle steps. Type is the set of operation types and $\tau: V \rightarrow \text{Type}$ is the mapping from an operation to its type. L is a set of module types and $\mu: P(\text{Type}) \rightarrow P(L)$ is a mapping from operation types to module types ($P()$ denotes the power set). $\mu(ts)$ returns the module types which can implement operations with a type from $ts \in P(\text{Type})$. $\text{area}: L \rightarrow \mathbb{R}$ returns the area of a module type. $n: L \rightarrow \mathbb{N}$ returns the number of selected modules of some type. $M(i)$ is the selected module set for $i \in \mathbb{N}$ cycles and $\text{lb_area}: \mathbb{N} \rightarrow \mathbb{R}$ returns the corresponding lower bound functional area.

$d: L \rightarrow \mathbb{N}^+$ describes the number of cycles a module type needs to execute an operation. $\text{dii}: L \rightarrow \mathbb{N}^+$ returns the data introduction interval (DII) of a module type. The DII is the minimal number of cycles required between the data arrivals for two executions. For simplicity it is assumed that the delays of a module are equal for all its operation types (the same holds for the DIIs); the presented method is however not restricted to the assumption made.

The ASAP (as soon as possible) value of an operation is the first cycle its execution could start, while the ALAP (as late as possible) value is the last cycle such that the execution of the operation must be completed afterwards. The operation execution interval OEI: $V \rightarrow P(T)$ describes for each operation the set of cycles in which it can be executed. *Initially* this set is equal to the cycles in the [ASAP, ALAP] interval assuming unlimited resources (so each operation is mapped on the module type with the smallest possible delay).

3. Lower bound module selection

The lower bound module selection is mainly based on relaxing the precedence relations of the operations in a DFG. The objective of module selection is to select a module set with minimal cost (i.e. minimal area) for a given time constraint and is NP-complete if the precedence constraints are taken into account [Garey79]. After relaxing the precedence constraints the module selection problem is 'easier' to solve. It can then be stated as finding the minimal set of modules needed to schedule the operations within their initial OEIs.

In this paper for the first time an approach will be presented which solves the relaxed module selection problem in polynomial time for *trivial* module libraries (i.e. libraries in which each operation type can be mapped on only one module type). The approaches of [Chen91], [Jain92] and [Shar93] (which handle only trivial libraries) do not guarantee that the operations can be scheduled within their initial OEIs with the selected module set and are therefore less accurate.

For unrestricted libraries the relaxed module selection problem is also NP-complete. The problem of finding the optimal module set for operations with fixed intervals on two types of modules can be identified as a special case of the relaxed module selection problem, and is proven to be NP-complete [Naka82]. However, we formulate a lower bound module selection problem as a very run time efficient mixed integer linear programming (MILP) problem. The approach tries to find bottlenecks in a DFG with respect to the module area needed, and formulates constraints for these bottlenecks. The object function of the MILP module selection problem is to minimize the function representing the total functional area:

$$\sum_{l \in L} (\text{area}(l) \times n(l)).$$

Only the variables denoting the number of selected modules have to be integral in the MILP formulation and the number of constraints is small. The run time efficiency of solving this MILP problem is therefore high and depends mainly upon the size of the library and not on the size of the DFG. Furthermore, a MILP solver can always come up with a feasible (but not necessarily optimal lower bound) solution just by ceiling the integer variables. In this way the approach can always come up with a solution within acceptable run times. In case of a trivial library the MILP problem can be solved in polynomial time because ceiling the integer variables always leads to the optimal solution.

In the following subsections different constraint sets to determine a lower bound module set for a given time constraint are discussed (see also [Timm93a] for a more thorough discussion of the module selection approach). The module set with the smallest area possible can be determined with the following set of constraints:

$$\forall_{t \in \text{Type}} : \sum_{l \in \mu(\{t\})} n(l) = 1.$$

3.1. Distribution constraints

Consider the example in figure 2 with a budget of 4 cycles. [Jain92] and [Chen91] estimate the number of modules needed from a trivial library, while assuming that all cycles are available to perform any operation. There are 4 additions and an adder can perform 4 additions within the time constraint, so the result of their

estimation is 1 adder. Those approaches only guarantee that the operations can be scheduled within the cycle budget but not within their execution intervals, which is a stronger relaxation of the original module selection problem than in our approach.

The execution intervals of v_1 , v_2 and v_3 show that 3 additions must be executed within 2 cycles, so at least 2 adders are needed. These 3 intervals form a so called 'distribution interval'. Distribution intervals can be derived from execution intervals as follows: if two execution intervals overlap they are joined into a new interval which is the union of the former two. The example shows that the notion of distribution intervals already leads to a more accurate estimation than the approaches of [Jain92] and [Chen91].

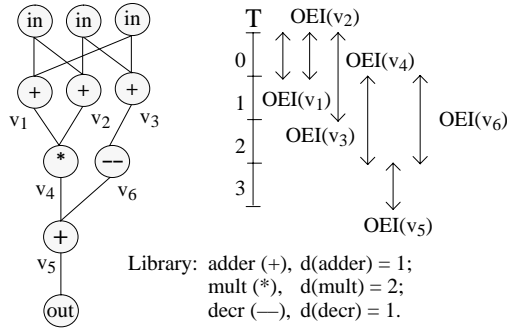


Fig 2: Example DFG, library and OEIs.

The corresponding constraints try to enforce the selection of sufficient module 'capacity' to perform the operations in the distribution intervals. For each (set of) operation type(s) only the interval with the highest average of operations per cycle step is of interest. This interval is called the *had-interval* (highest average distribution interval). All other intervals normally need equal or less module capacity and are therefore not considered in the MILP formulation.

To formulate the MILP constraints, the 'number of preliminary mappings' and the 'capacity' of a module type have to be defined. The capacity denotes the number of operations a module can execute within an interval. Because several module types can be selected for operations in an interval, the number of operations mapped on each type must be given. These numbers are preliminary, as the exact mappings are determined later on in the synthesis trajectory by a scheduler. The formal definitions and constraints are:

DEFINITION 1 [NUMBER OF PRELIMINARY MAPPINGS].

$m: L \times P(\text{Type}) \times \text{Type} \rightarrow \mathbb{R}$ is a function describing the number of preliminary mappings to a module type in a had-interval; $m(\text{alu}, \{+, *, -\}, +)$ is the number of additions mapped to the module type alu in the had-interval of the operation type set $\{+, *, -\}$.

DEFINITION 2 [CAPACITY OF A MODULE TYPE].

Let $l \in L$ and $e \in [T, T]$. $\text{cap}: L \times [T, T] \rightarrow \mathbb{N}$ is the function which describes the number of operations a module of type l can execute in an interval e :

$$\text{cap}(l, e) = \left\lfloor \frac{|e| - d(l) + \text{dii}(l)}{\text{dii}(l)} \right\rfloor.$$

Figure 3a shows the fifth order wave digital filter from [DeWi85]. During the execution of the fifth addition (operation 14), no other operation can be executing. Such operations can be identified by the fact that their initial OEIs do not overlap with the initial OEI of any other operation in case the time constraint is as tight as possible. As soon as the fifth addition is in the same distribution interval as any of its succeeding additions, there will be a 'hole' in that distribution interval which is as least as large as the minimal delay of a multiplication. During the hole no addition can be performed and the hole will occur after the fifth addition is executed. Such holes can be incorporated in the module selection by changing the capacity of a module type. The calculation of the capacity is changed by decreasing the available number of cycles $|e|$ in an interval $e \in [T, T]$ with the minimal number of cycles of the hole.

CONSTRAINT SET 1A.

Let $ts \in P(\text{Type})$, $t \in ts$ and $\text{no}(ts, t)$ be the number of operations $v \in V$ with $\tau(v) = t$ within the had-interval e of the operation type set ts . Then the following constraints must always be valid:

$$\forall_{ts \in P(\text{Type})} \forall_{t \in ts} : \sum_{l \in \mu(t)} m(l, ts, t) = \text{no}(ts, t),$$

$$\forall_{ts \in P(\text{Type})} \forall_{l \in \mu(ts)} : \sum_{t \in ts \mid t \in \mu^{-1}(l)} m(l, ts, t) \leq \text{cap}(l, e) \times n(l).$$

REMARKS. The variables $n(l)$ are the only integer variables in the above and the following constraint sets. If for a set of operation types not all types are present in the had-interval or there is no module type which can perform all these operation types, then the corresponding constraints have no impact at all and can be omitted.

The execution interval of each operation is of course an upperbound for its execution time, a condition not yet taken care of. The following constraint set takes this condition into account:

CONSTRAINT SET 1B.

Let $ts \in P(\text{Type})$, $t \in ts$, $i \in \mathbb{N}^+$ and let $\text{noo}(ts, t, i)$ be the number of operations $v \in V$, $\tau(v) = t$, $|\text{OEI}(v)| \leq i$ within the had-interval e of the operation type set ts . Then the following constraints must always be valid:

$$\forall_{ts \in P(\text{Type})} \forall_{t \in ts} \forall_{i \in [1, 2, \dots, |e|]} : \sum_{l \in \mu(t) \mid d(l) \leq i} m(l, ts, t) \geq \text{noo}(ts, t, i).$$

REMARKS. Many of the constraints of set 1b are superfluous and can be omitted when: the left hand side is equal to the left hand side of another constraint, while the right hand side of the other constraint is equal or larger; or when the right hand sides are equal, while the left hand side of the other constraint is a subset.

3.2. Fixed operation constraints

For operations which always occupy a module in a certain cycle step new constraints can be derived. As can be seen in figure 2, the operations v_1 and v_2 will always be scheduled in cycle 0, so there are at least two adders needed. The constraint set describing fixed operations can be formalized as:

CONSTRAINT SET 2.

Let $ts \in P(\text{Type})$, $i \in \mathbb{N}^+$ and $\text{nof}(ts, i)$ be the maximal number of operations $v \in V$, $\tau(v) \in ts$, $|\text{OEI}(v)| \leq i$, which always occupy a module in the same cycle step. Then the following constraints must always be valid (many constraints can be omitted in the same way as constraints of set 1b):

$$\forall_{ts \in P(\text{Type})} \forall_{i \in [1, |T|]} : \sum_{l \in \mu(ts) \mid d(l) \leq i} n(l) \geq \text{nof}(ts, i).$$

3.3. Path delay constraints

The previous constraints did not consider whether the minimal delay of each path in a DFG could be within the time constraint in case of an unrestricted library. The critical paths are those which could become the path with the largest minimal delay. Only these critical paths have to be considered in the respective constraints, and they can be determined efficiently by breadth first search. The corresponding constraints are:

CONSTRAINT SET 3.

Let $k: L \times \text{Type} \rightarrow [0, \dots, 1]$ be a help function, C the set of critical paths, and $\text{nop}(c, t)$ the number of operations $v \in V$, $\tau(v) = t$ within the path $c \in C$. Then the following constraints must be valid:

$$\forall_{c \in C} : \sum_{t \in \text{Type}} \sum_{l \in \mu(\{t\})} (k(l, t) \times d(l) \times \text{nop}(c, t)) \leq |T|,$$

$$\forall_{t \in \text{Type}} : \sum_{l \in \mu(\{t\})} k(l, t) = 1,$$

$$\forall_{t \in \text{Type}} \forall_{l \in \mu(\{t\})} : k(l, t) \leq n(l).$$

3.4. Additional constraints

For each time constraint the lower bound functional area will be larger than or equal to the lower bound functional area of the constraint plus one cycle (see figure 1). So the following constraint must be valid:

$$\text{lb_area}(i + 1) \leq \text{lb_area}(i), \quad i \in \mathbb{N}^+.$$

Although the constraint does not change the module selection result, it can decrease the search space of a MILP solver. During a branch-and-bound process the solver does not have to investigate module sets which violate this constraint, which can lead to considerable run time efficiency improvements.

4. Lower bound cycle budget estimation

The cycle budget estimation is based on the execution interval analysis of [Timm93b]. In that approach the *initial* OEIs (which were calculated under the assumption of unlimited resources) are reduced under influence of the (selected) set of resources. The reduction of the execution intervals prunes the search space of succeeding synthesis steps without limiting their solution space. The reduction is mainly achieved by finding the so called irreducible components in a bipartite graph matching formulation. In the following subsections the different aspects of the approach will be discussed briefly (see [Timm93b] for a more thorough discussion). An improvement on the run time complexity of the method will also be given.

4.1. Determination of MEIs

Operations must be scheduled within their OEI, so these intervals identify the maximum *freedom* of operations. Similarly module execution intervals (MEIs) identify the maximum freedoms of the module set for a (set of) operation type(s):

DEFINITION 3 [MODULE EXECUTION INTERVAL].

MEI: $P(M(i)) \times P(\text{Type}) \rightarrow P(T)$ describes a set of cycles in which *some* module $m \in M(i)$ applicable for the operation type set $ts \in P(\text{Type})$ (i.e. $\mu^{-1}(m) \cap ts \neq \emptyset$) has to perform *some* operation $v \in V$ with $\tau(v) \in ts$.

The MEIs are partially ordered based on their start and end cycles: the first MEI denotes the interval in which the first execution of an operation must take place, the second MEI the interval in which the second execution must take place, and so on. For the i^{th} MEI, the earliest possible start cycle and the latest possible end cycle of the i^{th} execution of an operation have to be calculated. Notice that MEIs are not always related to specific modules, while OEIs are related to specific operations.

For the determination of the earliest possible start cycles of MEIs we use the following algorithm which resembles the lower bound algorithm of [Rim92]. Let $\text{op}(ts) = \{v \in V \mid \tau(v) \in ts\}$ be ordered by increasing ASAP, let $\text{mods}(ts) = \{m \in M(i) \mid \mu^{-1}(m) \cap ts \neq \emptyset\}$ be the list of applicable modules ordered by increasing DII, let $\text{MEI}(i)$ be the start of the i^{th} MEI, and let $\text{start}(m)$ be the first cycle in which module $m \in \text{mods}(ts)$ can start a new execution. The next algorithm calculates in a left edge manner the correct start cycles

of the MEIs. In the function 'select' the module with the smallest DII among the earliest available modules is chosen to map an operation on. In the function 'new_start' possible swaps between faster and slower modules are incorporated by allowing a module m to be available again in less than $dii(m)$ cycles (see for more details [Timm93b]).

```

for (i := 1 to |mods|) →
  start(mods[i]) := 0;
for (i := 1 to |op|) →
  m := select(ASAP(op[i]));
  MEI(i) := max {ASAP(op[i]), start(m)};
  start(m) := new_start(m, MEI(i));

select (asap)
{
  choice := mods[1];
  for (j := 1 to |mods|) →
    k := mods[j];
    if (start(k) ≤ asap) →
      return k;
    if (start(k) < start(choice)) →
      choice := k;
  return choice;
}

new_start (m, MEI)
{
  for (j := 1 to |mods|) →
    k := mods[j];
    if (k = m) →
      return MEI + dii(m);
    last := last time module k started;
    if (start(m) ≤ last < MEI ∧
      last + dii(m) ≤ MEI + dii(k)) →
      return MEI + dii(k);
}

```

The end cycles of the MEIs can be determined similarly as the start cycles by applying the algorithms on the graph obtained by reversing the edges of G . The first execution of an operation must be completed at the first (i.e. earliest) end cycle. The first MEI is therefore determined by the first start and the first end cycle, the second MEI by the second start and end cycle, and so on.

An upper time bound for the determination of the end cycles is needed of course. From the start cycles of the MEIs, a *first* lower bound on the number of cycles can be derived. The approach continues by determining the end cycles of the OEIs and MEIs using the first bound. During this determination it can happen that the number of cycles within a MEI turns out to be smaller than the minimal delay of any applicable module $m \in \text{mods}(ts)$. In that case the first bound on the number of cycles and the previously determined end cycles were not correct and must be increased. The wrong MEI is updated and annotated with the increase. After all MEIs are determined, the wrong end cycles and the lower time bound are updated by traversing the MEIs in the reverse order in which their end cycles were calculated. During this traversal the annotated increases are incor-

porated in the end cycles of the OEIs and MEIs. If during the remaining analysis the lower cycle bound turns out to be incorrect, then the bound has to be incremented until it can possibly be correct.

4.2. Bipartite graph matching formulation

The remaining lower cycle bound estimation is based on a bipartite graph matching formulation. In figure 3a, the fifth order wave digital filter from [DeWi85] is presented with a lower cycle bound of 21 cycles and a set of resources consisting of 1 multiplier and 2 adders. For the multiplications the *initial* OEIs under the assumption of unlimited resources are given in figure 3c. There are 8 multiplications, so there are exactly 8 MEIs in which an applicable module must perform a multiplication. The multiplications have to be executed within 16 cycles (between cycle 4 and cycle 19). As the multiplier has a dii of 2 and the 8 operations have to be scheduled within 16 cycles, there are 8 (non-overlapping) MEIs of 2 cycles to perform the multiplications. Figure 3c shows these MEIs, which start every 2 cycles from cycle step 4 onward.

Undirected bipartite graphs $G_B(ts) = (N, A)$, $ts \in P(\text{Type})$ can be derived from the DFG and constraints with the following definitions and characteristics. The set of nodes $N = O \cup R$ with $O \cap R = \emptyset$, $|O| = |R|$, and the set of edges $A \subseteq O \times R$. The left set of nodes $O = \{v \in V \mid \tau(v) \in ts\}$, while the right set R is associated with the corresponding MEIs. There is an edge $(n_1, n_2) \in A$ if and only if operation n_1 could be scheduled within the MEI of n_2 (see figure 3c). The OEI or MEI of any node $n \in N$ can not be larger than the union of execution intervals of its adjacent nodes. For each feasible schedule there exists a corresponding *complete* matching, i.e. a matching with maximum cardinality $|O| = |R|$. If a complete matching is not possible, then the lower cycle bound must be incremented.

The OEIs can be reduced by identifying the edges in the bipartite graphs which can *never* belong to a complete matching. These edges can be deleted without limiting the solution space of a scheduler. The OEIs which were connected to the deleted edges can be reduced, as OEIs can not be larger than the union of MEIs of their adjacent nodes. The irreducible components of a bipartite graph are defined as the subgraphs induced by the edges which can be part of a complete matching. The irreducible components, and therefore the edges which can be deleted from a bipartite graph, can in the general case be derived in $O(|V|^{1/2} \cdot |A|)$ [Sang76].

The edges (15, [8,9]), (25, [8,9]), (18, [12,13]), (28, [12,13]), (22, [14,15]), (22, [16,17]), (42, [12, 13]), (42, [14, 15]) and (42, [16, 17]) in figure 3c do not belong to irreducible components and can be deleted. Clearly the analysis reduces the OEIs of the operations which were connected to these edges. Multiplication 22 remains only connected to the right node with the MEI

(a)



(b) Lower time bound: 21 cycles.
Resources: 1 multiplier (d=dii=2);
1 slow adder (d=dii=2);
1 fast adder (d=1).

(c) Initial bipartite graph matching formulation for the multiplications:

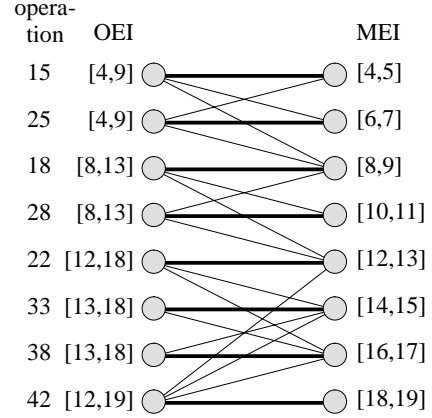


Fig 3: Fifth order wave digital filter from [DeWi85] with bipartite graph matching formulation.

[12,13], i.e. this multiplication is fixed and must be scheduled in the first two cycles of its initial OEI. The operations 15 and 18 are therefore also fixed to the first two cycles of their initial OEIs. When edges are removed from a bipartite graph and this removal leads to reducing the number of cycles in an OEI, a new 'run' can be started to reduce the OEIs even more. It was shown in [Timm93b] that the number of these runs is at most $O(|M| \cdot |V|^2)$ for one time bound, but in practice the algorithm already stops after a few runs.

In the approach of [Timm93b] the total complexity of determining a bipartite graph G_B and its irreducible components is $O(|V|^2 + |V|^{1/2} \cdot |A|)$, with $|A| \leq |V|^2$. The term $O(|V|^{1/2} \cdot |A|)$ is due to the derivation of the irreducible components which consists of two steps [Sang76]. In the first step a complete matching is determined which is $O(|V|^{1/2} \cdot |A|)$ in the general case [Hopc73]. In the second step the bipartite graph is directed by giving the edges in the matching a direction opposite to the direction of the edges which are not in the matching. Nodes which are incident with the same edge in the matching are then taken together with that edge to form a new node. The edges which connect a strongly connected component in this new directed graph form the edges of the corresponding irreducible

components of the original bipartite graph. These strongly connected components can be determined in $O(|V| + |A|)$ using depth first search.

The first step of finding a complete matching can be done in $O(|V| \cdot \lg |V|)$ instead of $O(|V|^{1/2} \cdot |A|)$ using the following algorithm. Let $op(ts) = \{v \in V \mid \tau(v) \in ts\}$ be ordered by increasing ASAP, let $latest(i)$ be the latest moment the operation in the i^{th} MEI can start (i.e. $latest(i)$ is the end cycle of the i^{th} MEI minus the minimal delay), let $op2$ be an initially empty list of operations and which is kept ordered by increasing ALAP, and let BM be the (initially empty) matching.

```
j := 1;
for (i := 1 to |op|) →
  while (ASAP(op[j]) ≤ latest[i]) →
    insert op[j] correctly in op2;
    j := j + 1;
  if ((op2[1], ith MEI) ∈ A) →
    insert (op2[1], ith MEI) in BM;
    delete op2[1] from op2;
  else
    no complete matching possible;
```

THEOREM.

The greedy algorithm above produces a complete matching if one exists.

PROOF.

The first operation $v_1 \in \text{op}(ts)$ to be selected for the 1st MEI is the operation with the smallest ALAP which can be scheduled within that MEI. Suppose a complete matching exists in which the 1st MEI is not matched with v_1 but with $v_2 \in \text{op}(ts)$, and in which v_1 is matched with the j^{th} MEI, $j \in \{2, 3, \dots, |\text{op}(ts)|\}$. Then there also exists a complete matching in which v_1 is matched with the 1st MEI and v_2 with the j^{th} MEI. Because v_1 can start in the 1st and the j^{th} MEI, $\text{ALAP}(v_1) \leq \text{ALAP}(v_2)$, the start of the j^{th} MEI is larger or equal to the start of the 1st MEI, and v_2 can start in the 1st MEI, there is also an edge between v_2 and the j^{th} MEI in the bipartite graph. So if there exists a complete matching, then there also exists a complete matching which starts with a greedy choice. Moreover, once the greedy choice of the first matching has been made, the problem reduces to finding a complete matching for the remaining nodes. By induction the greedy choice at every step of the algorithm produces a complete matching if one exists.

The $O(|V| \cdot \lg |V|)$ complexity of the algorithm above is due to the ordering of the lists of operations. With this new algorithm the total complexity of determining a bipartite graph G_B and its irreducible components becomes $O(|V|^2)$ instead of $O(|V|^2 + |V|^{1/2} \cdot |A|)$.

4.3. Additional analyses

Besides the determination of irreducible components of bipartite graphs, some other analyses can be performed to enhance the result of the lower bound cycle budget estimation.

Because of the 'hole' after the fifth addition in figure 3a (see section 3.1), the difference between the start (and end) cycles of the fifth and sixth MEIs of the additions must differ at least 2 cycles. Furthermore, the first five additions have to be executed before any other operation can be executed, so these additions must be mapped on the first five MEIs of the additions.

If an OEI is small enough, then the corresponding operation will always occupy a module in some specific cycle(s) independent of the chosen schedule (see also section 3.2). If in a cycle the number of such operations is equal to the number of modules, then all other operations cannot be scheduled in that cycle.

If for an edge $(n_1, n_2) \in A$: $\text{MEI}(n_2) \subseteq \text{OEI}(n_1) \wedge (\forall n_i: (n_1, n_i) \in A: \text{MEI}(n_2) \supseteq \text{MEI}(n_i))$, then the left node (or operation) n_1 can be assigned to the right node n_2 without limiting the solution space of a scheduler. All edges connected to these two nodes in the bipartite graph except for the edge between them can be removed.

5. Reviewing infeasible module sets

The cycle budget estimation is a partial check for the feasibility of the derived module sets. If the lower bound estimation of the cycle budget exceeds the number of cycles for which the module set was derived, then the module set cannot be correct. In that case it can be tried to detect why the module set cannot be feasible and to determine a new and better lower bound estimation of the module set with minimal area.

5.1 Detection in case of initial OEIs

Consider an *initial* bipartite graph matching formulation $G_B(ts)$, $ts \in P(\text{Type})$ with the *initial* OEIs and MEIs (i.e. no execution interval reduction has yet taken place), while the lower bound cycle budget estimation equals the time constraint for which the module set was derived. If in such a case no complete matching can be found, then there was not enough module capacity selected to be able to schedule the operations within their initial OEIs. In case of a trivial module library extra modules of the type $\mu(ts)$ can be added until a complete matching is found. The maximum number of such additions is $O(|V|)$, so the relaxed module selection problem formulated in the beginning of section 3 can be performed in polynomial time for a trivial module library.

In case of an unrestricted library one can not just add some module $m \in \mu(ts)$ and guarantee that the resulting module set is a lower bound estimation of the module set with minimal area. The reason that the module set is not correct can then be interpreted as follows. The distribution intervals try to detect local 'concentrations' of operations. If a module set is not correct then there exists a local 'concentration' of operations within a distribution interval which has not been detected and for which there is not enough module capacity selected. During the algorithm for finding a complete matching (see section 4.2), some MEI will not be matched and the algorithm stops. All operations which could have been matched with that MEI are matched with previous MEIs, and the MEI denotes a 'low' in the distribution of operations. If all the operations which could be scheduled within that MEI are deleted from the distribution interval, the interval will be split into two (or more) new intervals, and the local concentration is expected to be 'isolated' in one of them. That interval will then become the new had-interval, and a renewed module selection will lead to an improved lower bound estimation of the optimal module set.

5.2 Detection in case of reduced OEIs

If a complete matching can not be found during a second or later run of the execution interval reduction, then the module set can not be feasible because of the violation of some precedence constraints. A new lower

bound module set can then be derived by adding new constraints forcing to select a different set with a total area equal to or larger than the previous selection.

Let $M_p(i)$ be a previous module set which is detected as infeasible for $i \in \mathbb{N}$ cycles, let $area_p(i)$ be the corresponding lower bound module area, let $n_p(l)$ be the corresponding number of selected modules of type $l \in L$, let $n_{pos}(l) \in \mathbb{R}$ denote an increase in the number of modules of type l , let $n_{neg}(l) \in \mathbb{R}$ denote a decrease in the number of modules of type l , and let $y(l) \in \{0, 1\}$ be a help variable for module type l . The following constraints must be valid if $M_p(i)$ is not correct.

$$\begin{aligned}
 lb_area(i) &\geq lb_area_p(i). \\
 \forall_{l \in L} : \quad &n_{pos}(l) - n_{neg}(l) = n(l) - n_p(l); \\
 &n_{pos}(l) \leq |V| \cdot (1 - y(l)); \\
 &n_{neg}(l) \leq |V| \cdot y(l). \\
 \sum_{l \in L} (n_{pos}(l) + n_{neg}(l)) &\geq 1.
 \end{aligned}$$

These constraints can be added to the constraint sets of section 3 until a module set is selected which could be feasible according to the lower bound cycle budget estimation of section 4. These additional constraints due to a previous module set can be discarded as soon as the corresponding lower bound area becomes smaller than the lower bound area of the latest module selection.

6. Experiments and results

The area–delay curve prediction approach presented in this paper has been implemented in the NEAT system which is written in C++. A public domain solver ('lp_solve', which is available by anonymous ftp from ftp.es.ele.tue.nl) has been used for solving the MILP problems. The solver uses the simplex algorithm with sparse matrix methods for the linear programming part and branch–and–bound techniques for the integer part of the tool. In this section results of the approach on two benchmarks are given (the tests were run on a HP9000/755 workstation). The benchmarks are: WDEL F, the fifth order wave digital filter from [DeWi85] (see figure 3a) and FDCT, a fast discrete cosine transform which originated from [Mall90] (see figure 4). FDCT contains in contrast to WDEL F a lot of parallelism which can be reduced significantly when the number of available cycle steps increases.

In table 1, 2 and 3 different module libraries have been given. Table 1 gives a trivial module library which is used in most papers. Table 2 gives a extended module library which has already been used in [Timm93a], while the module library of table 3 with the largest range of delays originated from [Ishi91]. In table 4 and 5 the results for WDEL F for the libraries 1 and 2 are

given, while in figure 6 the results for WDEL F for the libraries 2 and 3 are given. In table 6 and 7 the results for FDCT for the libraries 1 and 2 are given, while in figure 5 the results for FDCT for the libraries 2 and 3 are given. In table 8 an overview of the results is given.

The figures 5 and 6 show that there is hardly any difference between the lower bound estimation and the optimal area–delay curve. Table 8 also shows that the estimations are really accurate and that the average CPU times are small. However keep in mind that the approach uses a MILP solver which can lead in some cases to unacceptable run times. This can be avoided by accepting intermediate (but not necessarily optimal lower bound) solutions to a MILP problem by ceiling the integer variables.

Table 1: Library 1 (trivial library).

module type	area	delay in cycle steps	operations		
			*	+	–
mult	144	2	x		
alu1	16	1		x	x

Table 2: Library 2 (extended library).

module type	area	delay in cycle steps	operations		
			*	+	–
mult	144	2	x		
alu1	16	1		x	x
sub1	15	1			x
add1	15	1		x	
alu2	9	2		x	x
sub2	8.5	2			x
add2	8.5	2		x	

Table 3: Library 3 (extended library from [Ishi91]).

module type	area	delay in cycle steps	operations		
			*	+	–
mpy1	256	1	x		
mpy2	32	16	x		
mpy3	2	256	x		
add1	16	1		x	x
add2	5	4		x	x
add3	2	16		x	x

Table 4: WDEL F results, lib. 1.

Table 5: WDEL F results, lib. 2.

<i>/T/</i>	<i>estimated area</i>	<i>optimal area</i>	<i>/T/</i>	<i>estimated area</i>	<i>optimal area</i>
17	480	idem	17	480	idem
18 to 20	320	idem	18 to 19	320	idem
21 to 27	176	idem	20	312.5	320
28 to	160	idem	21 to 27	168.5	idem
			28 to 53	160	idem
			54 to	152.5	idem

Fig 4: Fast discrete cosine transform (FDCT).

Table 6:
FDCT results, lib. 1.

<i>T</i>	<i>estimated area</i>	<i>optimal area</i>
8	1216	idem
9	1200	1216
10	768	784
11 to 12	624	idem
13	608	idem
14 to 17	464	idem
18 to 25	320	idem
26 to 33	304	idem
34 to	160	idem

Table 7:
FDCT results, lib. 2.

<i>T</i>	<i>estimated area</i>	<i>optimal area</i>
8	1212	idem
9	1198	1212
10	766	780
11	615	622
12	615	idem
13	606	idem
14 to 17	462	idem
18	319	320
19 to 25	312.5	idem
26 to 33	304	idem
34 to 51	160	idem
52 to	153	idem

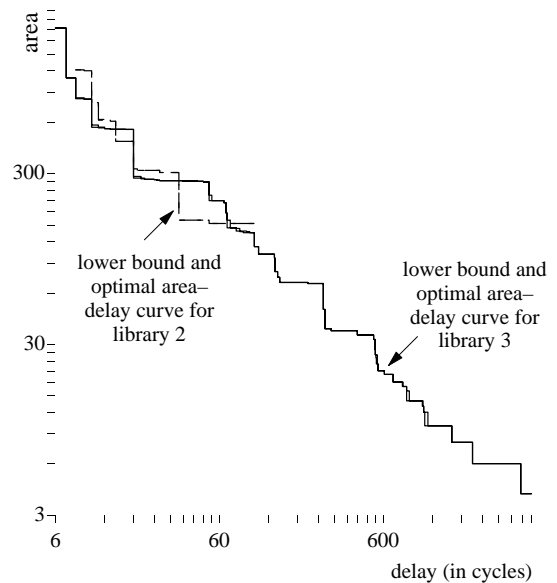


Fig 5: FDCT results

In [Ishi91] module sets are determined for only twelve different time constraints of WDELFF and for these module sets the smallest possible cycle budgets are determined. Six of the derived module sets of [Ishi91] were equal to the lower bound estimation and were therefore optimal. One other module set was optimal but not equal to the lower bound estimation. The remaining five module sets derived by the scheduling approach of [Ishi91] were not optimal. Only five of the twelve determined cycle budgets were equal to the lower bound estimation and were optimal. The other cycle budget estimations were the estimations for the seven smallest module sets and were not optimal.

[Rim92] and [Shar93] report lower bound cycle budget estimations for the module sets of table 4. One of the four estimations reported by [Rim92] and [Shar93] is not exact. Their estimations derive a lower bound of 27 cycles for the smallest module set, while the exact bound is 28 cycles (which is derived by our approach). The approaches of [Rim92] and [Shar93] also did not generate the module sets of table 4 by means of a lower bound estimation.

Conclusions

In this paper a unified approach of lower bound functional area and cycle budget estimations has been

Table 8: Overview of results.

	<i>range of time constraints (in cycles)</i>	<i>range of functional area</i>	<i>percentage of wrong estimations</i>	<i>largest absolute difference</i>	<i>largest relative difference</i>	<i>average CPU time per time constraint (in sec)</i>
WDELf with library 1	17 – 28	160 – 480	0%	0	0%	0.17
WDELf with library 2	17 – 54	152.5 – 480	2.63%	7.5	2.34%	0.25
WDELf with library 3	14 – 2144	4 – 560	2.30%	11	16.67%	0.21
FDCT with library 1	8 – 34	160 – 1216	7.41%	16	2.04%	0.12
FDCT with library 2	8 – 52	153 – 1212	8.89%	14	1.79%	0.22
FDCT with library 3	6 – 4128	4 – 2128	3.47%	17	16.67%	0.91

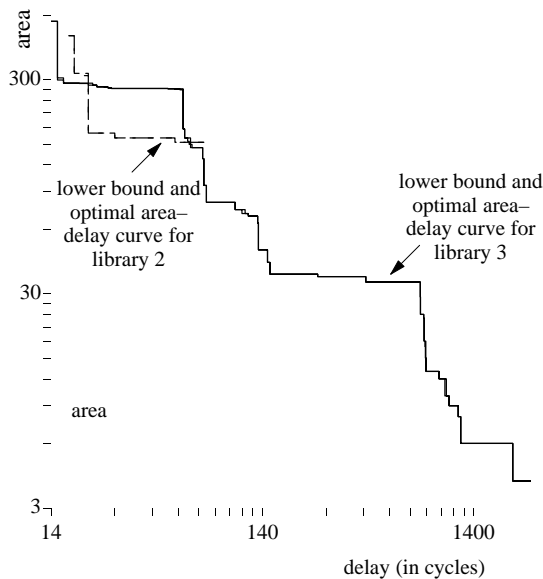


Fig 6: WDELf results

presented to predict the area–delay characteristics of a design at system level. The approach is implemented in the NEAT system and results for well-known benchmarks show fast running times and very accurate estimations. The approach leads in fact to the most accurate estimations reported to date. Because the estimations are so accurate, they can be used to measure the quality of a final design and to decrease the search space of a high-level synthesis system. The lower bound module sets are therefore used as a starting point in the synthesis trajectory of the NEAT system.

References

[Chen91] L.-G. Chen and L.-G. Jeng, "Optimal Module Set and Clock Cycle Selection for DSP Synthesis", Proc. ISCAS-91, pp. 2200–2203, 1991.

[Mall90] D.J. Mallon and P.B. Denyer, "A New Approach to Pipeline Optimization", Proc. EDAC '90, pp. 83–88, 1990.

[DeWi85] P. DeWilde, E. Deprettere and R. Nouta, "Parallel and Pipelined VLSI Implementations of Signal Processing Algorithms", in S.Y. Kung, H.J. Whitehouse and T. Kailath, VLSI and Modern Signal Processing, Prentice Hall, pp. 258–264, 1985.

[Eijnd92] J.T.J. van Eijndhoven and L. Stok, "A Data Flow Graph Exchange Standard", Proc. EDAC '92, pp. 193–199, 1992.

[Fleu93] H. Fleurkens, "Interactive System Design in ESCAPE", Proc. IEEE Int. Workshop on Rapid System Prototyping, pp. 108–113, 1993.

[Garey79] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, San Francisco, 1979.

[Hopc73] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", SIAM J. Comput., Vol. 2, No. 4, 1973.

[Ishi91] M. Ishikawa and G. DeMicheli, "A Module Selection Algorithm for High-Level Synthesis", Proc. ISCAS-91, pp. 1777–1780, 1991.

[Jain92] R. Jain, A.C. Parker and N. Park, "Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs", IEEE Trans. on CAD, vol. 11, no. 8, pp. 955–965, august 1992.

[McFa90] M.C. McFarland, A.C. Parker and R. Camposano, "The High-Level Synthesis of Digital Systems", Proc. of the IEEE, vol. 78, no. 2, pp. 301–318, 1990.

[Naka82] K. Nakajima, S.L. Hakimi and J.K. Lenstra, "Complexity results for scheduling tasks in fixed intervals on two types of machines", SIAM J. Comput, no. 11, pp. 512–520, 1982.

[Rim92] M. Rim and R. Jain, "Estimating Lower-Bound Performance of Schedules Using a Relaxation Technique", Proc. ICCD '92, pp. 290–294, 1992.

[Sang76] A. Sangiovanni-Vincentelli, "A Note on Bipartite Graphs and Pivot Selection in Sparse Matrices", IEEE Trans. Circuits & Systems, vol. CAS.23, no. 12, pp. 817–821, 1976.

[Shar93] A. Sharma and R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applications", Proc. of the 30th DAC, pp. 355–360, 1993.

[Timm93a] A.H. Timmer, M.J.M. Heijligers, L. Stok and J.A.G. Jess, "Module Selection and Scheduling using Unrestricted Libraries", Proc. EDAC/EuroASIC '93, pp. 547–551, 1993.

[Timm93b] A.H. Timmer and J.A.G. Jess, "Execution Interval Analysis under Resource Constraints", Digest of technical papers of ICCAD-93, 1993.