Optimal Allocation and Binding in High-Level Synthesis

Minjoong Rim¹, Rajiv Jain¹ and Renato De Leone²
¹Department of Electrical and Computer Engineering
²Department of Computer Sciences
University of Wisconsin
Madison, WI 53706

Abstract
In this paper, we present an integer linear program (ILP) formulation for the allocation and binding problem in high-level synthesis. Given a behavioral specification and a time-step schedule of operations, the formulation minimizes wiring and multiplexer areas. This is the first time that an ILP model for minimizing multiplexer and wiring areas has been mathematically formulated and optimally solved. The model handles chaining, multi-cycle operations, pipelined modules, conditional branches and trades off wiring area with resource area.

1 Introduction
Automatic generation of RTL (register-transfer level) designs from a behavioral description is known as high-level synthesis. To synthesize a desired RTL design, several conflicting and interacting sub-problems have to be solved. These sub-problems are scheduling, module allocation, module selection, module binding, and multiplexer and register allocation. In this paper, we present a new technique for allocation and binding with an objective of minimizing wiring and multiplexing areas required for interconnecting the design. Allocation is the task of ascertaining the number of modules of each type, and the number of registers and multiplexers required for the design. Binding is a mapping of operations to modules and values to registers. The binding and allocation tasks are inter-dependent and also depend upon scheduling, interconnect costs and delays. The input to our allocation and binding formulation is a scheduled data flow graph. First, we formulate the problem as a zero-one integer linear programming (ILP) problem and then use existing ILP solution techniques to solve the formulation.

The synthesis problem was first formulated as a mixed ILP problem by Hafer and Parker [4]. Hafer's formulation, however, does not minimize the number of wires or multiplexers used in the design with the result that the final design may actually require large multiplexer and wiring areas. Furthermore, this formulation entails enormous runtime and is not practical. By decomposing synthesis into two sub-problems, namely, (i) scheduling, and (ii) allocation and binding, we can approach each part independently and quickly solve them optimally. Solving each individual sub-problem independently and combining the solutions, however, does not necessarily lead to a globally optimal solution. The assumption of breaking synthesis into these two sub-problems has been used by almost all synthesis systems found in literature [10]. The notable exceptions are [4] in which all sub-components are solved concurrently, and [3] in which the synthesis problem is solved by partitioning it into two sub-problems different from the above breakdown: the scheduling, module allocation and module binding problem and, the register and multiplexer allocation problem. Recently, an ILP model for performing scheduling, allocation and binding concurrently has been proposed by Gebotys [2]. This model, however, like the work by Hafer and Parker [4] ignores multiplexer and wiring area in the formulation and gives an inferior design. Hafer has recently proposed solution to the high-level synthesis problem with the objective of minimizing wiring and multiplexer area. Wiring comprises of a substantial part of chip real estate and one cannot over-emphasize the importance of minimizing it.

2 Problem Formulation
We assume point-to-point interconnection style. After binding, we can get a partial floorplan and global routing information and wires can be combined to form buses using techniques like those in [9]. Even when other bus structures are used, our method is expected to result in shorter buses and less interconnections between buses and resources. The model is initially developed for a non-pipelined design style. Chaining, multi-cycle operations, multi-cycle values, and pipelined modules are allowed. An important advantage of our formulation is that extensions for handling these features do not require additional constraints or variables. Thus, the formulation complexity does not grow with these generalizations. Extensions for functional pipelining are elementary and do not require any additional constraints or variables, as we will see later. We assume that all operations produce only one output. This assumption is made for simplicity of explanation and is relaxed later. Handling of DFGs with conditional branches and modules with functional pipelining have been successfully modeled and have been omitted from the paper due to the lack of space. Details can be found in [13]. The following terms will aid in the discussion. In the formulation, there is no difference between operations and values or between modules and registers. In the remainder of the paper operations and
values are collectively called tasks and modules and registers are collectively called agents. TaskType is a set of task types. AgentType is a set of agent types. Time is a set of time-steps. I is a set of tasks. J is a set of agents. \( I_t = \{ i \in I \mid t \text{ is active in time-step } t \} \) where \( t \in \text{Time} \). A task is said to be active if it is scheduled in that time step. A unit-cycle operation is active during only one time-step whereas a multi-cycle operation is active over several time-steps. A pipelined operation is active in all the first \( l \) time-steps starting from the time-step it is first scheduled in. The output of a pipelined task, however, is available only after the total delay of the task. Interconnection is a set of ordered triples \((i_1, i_2, t)\) each of which represents interconnection between the output of task \( i_1 \) and the \( t^{th} \) input of task \( i_2 \). If tasks \( i_1 \) and \( i_2 \) are operations, it represents chaining of operations. Notice that multi-cycle operations, chaining, and pipelined modules are considered by extracting proper input data. Even though in the formulation we do not explicitly discuss multi-cycle operations, chaining of operations or pipelined modules, they are taken care of. We write \( \text{type}_{ta}(t) \) to denote the type of task \( i \), \( \text{type}_{ag}(j) \) to denote the type of an agent \( j \).

The objective of allocation and binding is to minimize the overall cost (area) of an RTL design. That is,

\[
\text{minimize } \text{module area} + \text{register area} + \text{multiplexer area} + \text{wiring area}
\]

A minimum number of modules of each type and registers required by the implementation are determined by the schedule and the max-cut of the input description respectively. While solving the allocation and binding problem, module and register areas are assumed to be constants \(^1\), and the objective function minimizes wiring and multiplexer areas. In general, a large number of connections between resources (register and modules) requires large quantities of multiplexers and wires. Hence, first, we assume that the number of multiplexers can be reduced by minimizing the number of wires and use this as an objective function. A similar objective has been adopted in [11]. Minimizing wiring and multiplexer areas concurrently is described later. We now formulate our problem with an objective of minimizing wiring area.

\[
\text{Module and Register Binding: } \text{Let } x_{i,j} = 1 \text{ if task } i \text{ is assigned to agent } j \text{ and zero otherwise. If task } i \text{ cannot be performed by agent } j \text{ due to type conflict, we set } x_{i,j} = 0.
\]

Each task should be performed by at least one agent. This is enforced by,

\[
\sum_{j \in J} x_{i,j} \geq 1, \quad i \in I
\]

Wiring area may be reduced by having more than one agent perform the same task but this is very unlikely. Hence, we require each task to be assigned to exactly one agent, and

\[
\sum_{j \in J} x_{i,j} = 1, \quad i \in I
\]

Since an agent can perform at most one task in a time-step, we have

\[
\sum_{i \in I_t} x_{i,j} \leq 1, \quad j \in J, \quad t \in \text{Time}
\]

Constraints 2 and 3 represent a simple assignment problem.

**Wiring Area:** A wire is defined to be an interconnection between two different agents. It excludes multiplexer area and split wires at the inputs of the multiplexers. Let \( w_{j_1,j_2} = 1 \) if there exists a value transfer from agent \( j_1 \) to agent \( j_2 \) in any time-step. If \((i_1, i_2, t) \in \text{Interconnection}\) there is a data transfer from task \( i_1 \) to task \( i_2 \), and therefore there must be a wire from the agent executing task \( i_1 \) to the agent executing task \( i_2 \). This can be modeled by,

\[
w_{j_1,j_2} \geq x_{i_1,j_1} + x_{i_2,j_2} - 1, \quad j_1, j_2 \in J
\]

Wiring area can now be written as

\[
\sum_{j_1 \in J} \sum_{j_2 \in J} c_{j_1,j_2} w_{j_1,j_2}
\]

where \( c_{j_1,j_2} \) is the cost of connecting agent \( j_1 \) to agent \( j_2 \), and is the area of the wire. If preliminary floor-planning has already been done, then these values may be estimated from the floorplan and used in the equation. In the absence of a floorplan, an estimate of the average wire length can be obtained using techniques given in [8]. In this case, all \( c_{j_1,j_2} \)'s are equal and can be eliminated from the objective function.

The number of variables in the formulation can be reduced by taking into account type conflicts. Let \( G_k \) be the number of tasks of type \( k \) \((k \in \text{TaskType})\). Let \( F \) be the total number of agents, and \( F_k \) be the number of agents which can perform a task of type \( k \). The number of \( x_{i,j} \) decision variables is \( \sum_{k \in \text{TaskType}} G_k F_k \), and the number of \( w_{j_1,j_2} \) decision variables is at most \( F^2 \). \(^2\) Thus, at most \( (F^2 + \sum_{k \in \text{TaskType}} G_k F_k) \) binary decision variables are needed in the model. Note that the value of \( w_{j_1,j_2} \) is completely determined by the variables \( x_{i,j} \)'s (assuming \( c_{j_1,j_2} \)'s are positive). The complexity of the problem can be reduced by declaring variables \( w_{j_1,j_2} \) as non-negative real variables, and using mixed integer programming techniques with at most \( \sum_{k \in \text{TaskType}} G_k F_k \) binary decision variables.

**Functional Pipelining:** The formulation presented so far is aimed at non-pipelined designs but it can be extended to functional pipelining for a fixed number of stages.

\(^1\)Tradeoffs between active, wiring and multiplexer areas are discussed in [13].

\(^2\)In practice many \( w_{j_1,j_2} \) variables can be set to zero since certain agents cannot be connected.
easily extended to the pipelined design style as well. Let \( t \) be the initiation interval of the pipelined data path. Then, the operations in time steps \( t + n \times \tau \) \((n = 1, 2, \ldots)\) are executed simultaneously and cannot share modules. Consequently, a set of active tasks in time-step \( t \) is defined to be,

\[ I_t = \{ i \in I \mid \text{task } i \text{ is active in time-step } t + n \times \tau \} \quad t = 1, 2, \ldots, l. \]

The problem formulation itself remains unaltered.

**Multi-Output Operations:** So far we have assumed that an operation generates only one unique value. Many operations generate several values which are fed to more than one resources, for example, an addition operation generates a carry value and a sum of its two inputs. Such operations can be accommodated in the following simple manner. We redefine \textit{Interconnection} to be a set of ordered quadruples \((i_1, m, i_2, l)\) each of which represents interconnection between the \( m \)th output of task \( i_1 \) and the \( l \)th input of task \( i_2 \). If task \( i \) produces \( g \) outputs, then \( M_{i}(i) = \{1, 2, \ldots, g\} \). If task \( i \) is of type value, \( M_{i}(i) = \{1\} \). If agent \( j \) produces \( g \) outputs, then \( M_j(j) = \{1, 2, \ldots, g\} \). If agent \( j \) is of type register, \( M_j(j) = \{\} \).

Wiring area is now modeled as follows. Let \( w_{j_1, j_2, m, j_3} \) be 1 if there exists a value transfer from the \( m \)th output of agent \( j_1 \) to agent \( j_2 \) in any timestep. If \((i_1, m, i_2, l) \in \text{Interconnection} \), there is a data transfer from the \( m \)th output of task \( i_1 \) to task \( i_2 \), and there must be a wire from the \( m \)th output of the agent executing task \( i_1 \) to the agent executing task \( i_2 \). This is given by

\[ w_{j_1, j_2, m, j_3} \geq x_{i_1, j_3} + x_{i_2, j_3} - 1, \quad j_1, j_2 \in J \quad (6)\]

Wiring area can now be written as

\[ \sum_{j_1 \in J} \sum_{m \in M_{j_1}(j_1)} \sum_{j_3 \in J} c_{j_1, j_2, m} w_{j_1, j_2, m, j_3} \quad (7) \]

where \( c_{j_1, j_2, m} \) is the cost of connecting the \( m \)th output of agent \( j_1 \) to agent \( j_2 \).

**Multiplexer Area Minimization:** One assumption made in the research literature is that minimizing multiplexer area results in minimum wiring area as well (or vice-versa) \[11\]. One can easily construct an example where minimizing one does not necessarily minimize the other. In order to achieve an optimal design we need to minimize both multiplexer and wiring areas. We now establish constraints with the objective function for minimizing multiplexer area. These constraints, when combined with the constraints and objective function for wiring area, minimize both multiplexer and wiring areas.

First, we define a few terms. If task \( i \) is a \( g \)-ary operation, then \( \Lambda_i(i) = \{1, 2, \ldots, g\} \). For a task \( i \) of type value, \( \Lambda_i(i) = \{1\} \). If an agent \( j \) is of type \( g \)-input module, then \( \Lambda_j(j) = \{1, 2, \ldots, g\} \). If agent \( j \) is of type register, then \( \Lambda_j(j) = \{\} \).

Multiplexer area at an input port of an agent is assumed to increase linearly in the number of distinct inputs to the input port. Let \( z_{j_1, j_2, l} \geq 1 \) if there exists a transfer of a value from agent \( j_1 \) to the \( l \)th input port of agent \( j_2 \) and zero otherwise. Suppose that task \( i_2 \) is a register or a non-commutative operation. If \((i_1, i_2, l) \in \text{Interconnection} \), then there must be a connection from the agent executing task \( i_1 \) to the \( l \)th input port of the agent executing task \( i_2 \).

\[ z_{j_1, j_2, l} \geq x_{i_1, j_3} + x_{i_2, j_3} - 1, \quad j_1, j_2 \in J \quad (8) \]

For commutative operations we need a slightly more complicated model. The difficulty with a commutative operation stems from the fact that a value can go to any input port of the corresponding agent. We introduce a concept of \textit{pseudo input port}. Each input value arrives at a pseudo input port which is connected to each functionally equivalent input port of the agent. The mapping from pseudo-input ports to input ports is an assignment problem. Let \( s_{l, p, i} = 1 \) if there exists a connection between pseudo-input port \( l \) and input port \( p \) of commutative task \( i \). The constraints can be written as

\[ \sum_{l \in L_{i_2}(l)} s_{l, p, i} = 1, \quad p \in L_{i_1}(p) \quad (9) \]

\[ \sum_{l \in L_{i_1}(l)} s_{l, p, i} = 1, \quad l \in L_{i_2}(l) \quad (10) \]

for each commutative operation. Then,

\[ z_{j_1, j_2, l} \geq x_{i_1, j_3} + x_{i_2, j_3} + s_{l, p, i} - 2, \quad l \in L_{i_2}(l), \quad j_1, j_2 \in J \quad (11) \]

The multiplexer area is given by

\[ \sum_{j_1 \in J} \sum_{j_2 \in J} \sum_{i \in L_{a}(j)} d_{j_1, j_2, l} z_{j_1, j_2, l} \quad (12) \]

where \( d_{j_1, j_2, l} \) is the multiplexer cost corresponding to \( z_{j_1, j_2, l} \). The objective is to minimize a linear combination of expressions \( 5 \) and \( 12 \).

For some cases, the number of multiplexers for each port at an agent may be limited. For example, total delay of multiplexers at a module should not exceed some value due to timing constraints. This is modeled as

\[ \sum_{j_1 \in J} \sum_{j_2 \in J} z_{j_1, j_2, l} \leq f_{j_1, l} \quad (13) \]

\[ \textit{Examples and Results}\]

By convention adopted in literature \[6 \] \[11 \] \[12 \] input registers are read-only buffers and cannot be reused. Also, a multi-cycle value is restricted to be stored in one register over several time-steps \[7 \]. This restriction can be easily overruled in our formulation.
result in inferior designs, specially in the pipelined design style where multi-cycle values are generally stored in several registers. This restriction can be easily eliminated without any modification to our formulation by simply breaking a multi-cycle value into several single cycle values.

To verify the formulation, we synthesized several RTL designs. We chose the complex multiplication, differential equation, AR filter and EW filter from high-level synthesis literature. The results are summarized in Table 1. All solutions are optimal with respect to the objective function of minimizing number of wires in the design. These examples illustrate chaining of operations, use of pipelined modules, conditional branches and makes tradeoffs between wiring area and module and register area. Most of these generalizations are achieved without an increase in the number of variables or constraints. We have also synthesized several examples from literature to demonstrate the practicality of the approach. All solutions were optimal.

References