Some New Results in the Complexity of Allocation and Binding in Data Path Synthesis

C.A.Mandal[1] *P.P.Chakrabarti*[2] *S.Ghose*[2]

[1] Department of Computer Science
Brunel University
Uxbridge UB8 3PH, U.K.
[2] Department of Computer Science & Engineering
Indian Institute of Technology
Kharagpur 721302, INDIA

Abstract

In this paper we present some new results on the complexity of allocation and binding problems in Data Path Synthesis (DPS). We have considered are the port assignment problem for multi-port memories, the register-interconnect optimization problem (RIO) and the problem of formation of functional units. RIO is a major problem of DPS and we have examined several versions of it. The simplest case that we have considered is register optimization (RO) for straight line code which is solvable in polynomial time. The next more general case that we have considered is RIO for straight-line code (SRIO), a special case of RIO, which we have shown to be NP-hard. The most significant contributions of this work are results on the hardness of relative approximation of several problems of DPS. We have shown that the constant bounded relative approximation of PA for triple port memories and SRIO are both NP-hard.

Key Words: Allocation, NP-complete problems, Data Path Synthesis High-Level Synthesis

1 Introduction

Allocation and binding problems which have to be addressed for data path synthesis include *a*) functional unit (FU) formation, *b*) interconnect formation, *c*) memory allocation, and *d*) register optimization. Together these make up a major part of the allocation and binding problem whose complexity we examine in this paper.

The overall problem of allocation and binding is NP-hard because some of its subproblems are already known to be NP-hard. The NP-hardness of the register minimization problem is one of the oldest complexity results in this area. The register minimization problem has been formulated as a clique partitioning problem in [1] which is a standard NP-complete problem. The complexity of connectivity binding has been examined in [2] where results on two restricted cases have been derived. The first case is as follows. Given i) a set of registers R where $|R| = \rho$, ii) a set of operation types Φ, iii) a set of units U where $|U| = \mu$ and each unit performs a subset of the operations in Φ, iv) a state graph $G = (V, A)$ where $|V| = \nu$ and $|A| = \alpha$ and v) a state assignment $f_s : v \rightarrow s$ of vertices to states in S where $|S| = \sigma$, vi) a binding of arcs in A that extend across state boundaries to registers in R, $f_r : a \to r$, and vii) an integer N. The problem is stated as follows: Is there a mapping of the vertices in the state graph G to the units in U, $f_u : v \to u$, such that each unit is used at most once in any state $s \in S$ and that the number of connections between registers and units is less than or equal to N? The second problem instance analyzed is as follows: Given a state graph $G = (V, A)$ and a set of registers R, a set of function units U, $f_u : v \to u$, and a positive integer N, is there a binding $f_r : a \to r$, of arcs in A to registers in R such that the number of connections between registers and units is less than of equal to N? Both the cases have been shown to be NP-Complete. The general decision problem has also been proved to be NP-Complete.

In our study of the complexity of allocation and binding we consider several new problems which are as follows.

- The port assignment of multi-port memories (PA).
- Register-interconnect optimization (RIO).
- The problem of functional unit formation (FUF).

In our study we not only examine the complexity of optimally solving the above mentioned problems, but also seek to establish the complexity of finding an approximate solution wherever possible.

In general for a NP-hard optimization problem we do not expect to find a polynomial time algorithm to solve that problem optimally. The only known methods of obtaining optimum solutions (like branch an bound techniques) are exponential in time complexity. Data Path Synthesis (DPS) problems which occur in practice are usually so complex that enumerative approaches like branch and bound are ruled out in many situations. The alternative approach is to relax the requirement of finding optimal cost solutions. Designers are often satisfied with a fast (polynomial time) algorithm provided it guarantees some error bounds on the cost of the solution. Such algorithms are known as approximation algorithms. There are two well known types of approximation algorithms, viz. the absolute approximation algorithm (which guarantees that the solution obtained will not differ from the optimal by more than a fixed constant) and the relative approximation algorithm (which guarantees that cost of the solution obtained by the algorithm will not exceed, in the case of minimization, the cost of the optimal by a constant factor). Absolute approximation algorithms are usually difficult to find for most NP-complete problems. The second type of approximation algorithms, namely those of relative approximation, are obtainable for many problems where an absolute approximation algorithm is not obtainable. Relative approximation algorithms are available for many scheduling problems. For example, list scheduling guarantees that the solution obtained in the case of a single operation DAG is never more than twice the optimal schedule length.

We shall establish that for nearly all the allocation problems that we consider the problem of finding a relative approximation itself is NP-complete. Throughout this paper we shall use the following notation: for any problem X, (for which we require an optimal solution) X-R shall denote the problem of finding a solution whose relative error is bounded by a constant. X-R will also be referred to as a constant bounded relative approximation for X. When we say that X-R is NP-complete we mean that if there is a polynomial time approximate algorithm which guarantees a constant relative error bound for X, then P=NP. In this paper we show that not only are most allocation problems NP-complete but also that constant bounded relative approximation of several versions of the PA and RIO are NP-complete.

The organization of the paper is as follows. In section 2 we introduce the general node deletion problem. We show that this problem as well as its relative approximations are NP-hard. These results form the basis of most of the results derived later in this paper. We introduce the port assignment problem in section 3. These problems arise when multi-port memories are used as building blocks in data path synthesis [3]. The problem of PA for triple port memories is examined in section 4. For triple port memories we have proved that the port assignment problem and its absolute and relative approximations are all NP-hard. In section 5 we review the register optimization problem and introduce the register optimization problem for straight line code, which is known to be solvable in polynomial time [4]. In section 6 we generalize this problem to the register-interconnect optimization problem for straight line code (SRIO). This problem is naturally encountered for simple instances for DPS problems. Through this problem we prove that the interconnect optimization as well as its constant bounded relative approximation are NP-hard. The NP-hardness of SRIO is a significant result because the problem of register optimization for straight line code is solvable in polynomial time and the problem of general register-interconnect optimization is already known to be NP-hard.

In section 7 we examine a different problem, namely that of functional unit formation (FUF). The FUF problem comes up when the schedule of operations does not explicitly indicate the FU to which an operation should be mapped. This is often the case when specific FUs are not available at the time of scheduling. During allocation and binding it is then necessary to find a suitable mapping of the operations to the FUs. This mapping determines the capabilities and therefore the cost of the FUs. We prove in particular that the problem of minimizing the cost of the FUs as well as its absolute approximation are NP-hard.

2 General Node Deletion

The node deletion problems have been used, in this paper, to establish the complexity of many of the sub-problems encountered during allocation and binding. The node deletion problem (ND) [5] is as follows. *Given a graph $G(V, E)$, V being the set of vertices and E the set of edges, identify a set of nodes N_d for deletion such that the graph, after deletion of nodes in N_d, is bipartite.* A graph is said to be bipartite if it can be partitioned into two independent sets. A graph is bipartite if and only if it can be coloured using two colours, i.e. it is 2-colourable. For the node deletion problem N_d should be the smallest possible set of such vertices. The node deletion decision problem (NDD) answers the question whether the given graph can be rendered bipartite by the deletion of m vertices, $0 \leq m \leq |V| - 2$. It has been shown that both ND and NDD are NP-complete [5].

The general node deletion problem (GND) is formulated as follows. Given a graph $G(V, E)$, identify a set of nodes N_d for deletion such that the graph, after deletion of nodes in N_d, is k-colourable. N_d should be the smallest possible set of such vertices. The corresponding decision problem (GNDD) is to answer the question whether the given graph can be rendered k-colourable by the deletion of m vertices, $0 \leq m \leq |V| - k$. Such a decision problem will be represented as GNDD(m, k).

Theorem 1 *The general node deletion decision problem (GNDD) is NP-complete.*

Proof: For 2-colourability GNDD is NP-Hard because NDD is NP-Hard. For k-colourability, $k \geq 3$, GNDD is NP-Hard because the corresponding chromatic decision problem is NP-Hard. It is easy to write a polynomial time non-deterministic algorithm [5] to solve GNDD. □

The following corollary follows easily.

Corollary 2 *GND is NP-complete.*

The question of polynomial time approximate algorithms for GND is now examined. An approximation algorithm for GND with constant relative error bound would be one which guarantees $d_r \leq (1+\epsilon)d^\star$, for any instance of GND, where $d^\star$ is the number of vertices deleted by the optimal algorithm, d_r is the number of vertices deleted by the approximation algorithm for this approximation and $\epsilon > 0$ is a constant. We denote the problem of obtaining an approximation algorithm for GND whose relative error is bounded by a constant as GND-R. This is also referred to as the constant bounded relative approximation for GND. The following important result can be proved.

Theorem 3 *A constant bounded relative approximation for GND (GND-R) is NP-complete.*

Proof: Suppose that a polynomial time algorithm exists which guarantees that the relative error in the approximate solutions to GND is bounded by a constant. Let it be A_r. Let $G(V, E)$ be an instance of GND, such that the graph is k-colourable. Therefore $d^\star = 0$. This requires that A_r should report $d_r = 0$. If this is so, then A_r could be used to solve the chromatic decision problem (CDP) [5] in polynomial time. CDP being NP-complete, A_r will be such a polynomial time approximate algorithm for GND only if it is also an optimal algorithm for CDP. Thus GND-R must be NP-complete. □

The particular case for GND to ensure that a given graph will be made three colourable will be referred to as GN3D. The following result regarding a constant bounded relative approximation for GN3D follows from theorem 3, since the three colour decision problem is NP-hard. We shall use this result several times in the rest of this paper.

Corollary 4 *GND-R for* 3*-colourability, GN3D-R, is NP-complete.*

3 Port Assignment

The variables that are used to specify a behaviour need to be implemented as storage elements. These may be clustered into memory modules of one, two or three ports. Some work on port assignment has been reported in [6]. At this level of abstraction it is permissible to view inputs and outputs of components, as well as ports of memories as single points in the circuit. A point in a circuit is said to access a memory if it transfers data to or from a cell of the memory. Given a set of registers being placed in the memory along with the permissible number of ports and their capabilities, and the set of accesses to these registers; it is necessary to assign the accessing points (in the circuit) to the memory ports so that all the access will be satisfied. The assignment should be made in such a way that the resulting interconnect overhead is minimized. The assignment process is explained with a simple example.

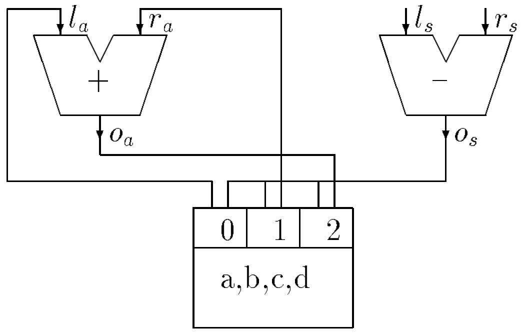

Figure 1: Connections to a three port memory.

Example 1 Consider the transfers given below.

1. $a = b + c$;
2. $q = c + d, \quad b = p - q$;
3. $d = a + r, \quad c = p - r$;
4. $a = p + c, \quad b = q - r$;

Suppose a, b, c, d, p, q, r and s are registers, of which only a, b, c and d are to be placed in the same memory. Assume that three ports are permitted, and the ports are labeled $0, 1$ and 2 respectively. It will be noted that at most three accesses are made to the memory in any time step. Suppose that an adder and a subtracter are used. Let the adder inputs be labeled l_a and r_a while the adder and subtracter outputs be respectively labeled o_a and o_s. It will be observed that l_a, r_a, o_a and o_s are the only four points accessing the memory in the various control steps. They need to be assigned to the ports suitably. Consider the assignment where l_a, r_a and o_a are mapped on ports $0, 1$ and 2 respectively, and o_s is mapped to all the ports $0, 1$ and 2. All the transfers can be satisfied using this assignment. The connections are illustrated in figure 1. It will be noted that a total of six switches will be required at the ports of the memory shown. □

It will be noted that a point in the circuit which must access a member of the memory under consideration, will be connected to at least one of its ports. To reduce interconnect overhead port assignment (PA) should be done so that the minimum number of points is connected to more than one port. When a point that reads from the memory is connected to $k \quad (k > 0)$ ports of the memory, each connection has to be switched through a multiplexer channel. Similar multiplexing is required when multiple sources are connected to a write port of a memory. For the port assignment suggested in example 1, o_s is mapped to all the three ports and so its connectivity is three, as against the desired level of one.

An algorithm that finds an optimal PA should lead to an interconnection that requires the least number of switches for multiplexing. In this section we shall deal with the PA problem for triple port memories. We shall first introduce the problem. We shall identify a special case of the corresponding PA problem to derive the complexity results.

The PA problem may have several variations, depending on the number and type of the ports. The ports may be uniform , being, read/write (rw) or purely read (r); or may have arbitrary capability, i.e. rw, r or w. PA for a single port is trivial. The cases for three ports with uniform capability is considered here. The complexity results for this case will help us to derive results for the complexity of the SRIO problem.

4 Memories with Three Uniform Ports

Given a set of registers which have been placed in a particular memory with three uniform ports, it is necessary to assign the circuit points accessing the memory such that a) all the accesses in each control step are satisfied and b) the cost of switches for interconnection to the ports is minimized. This problem will be referred to as PA3U. First a relaxed formulation PA3UA1 based on GND is presented in section 4.1. PA3UA1 will be proved to be as hard as GND. PA3UA1 will then be used to derive complexity results for PA3U.

4.1 PA3UA1 and Its Relationship with GN3D

We now consider a relaxed formulation of PA3U, to be referred to as PA3UA1, which is as follows. *The accesses to the memory in question are only read accesses, the points that read from the memory do not read from or write to any other place in the circuit and these points are connected to exactly one port or to all the three ports of the memory.* The first condition implies that multiplexers are required only at the circuit points and not at the ports. The second assumption ensures that the only lines incident at these circuit points are those coming from the memory ports. If a point is connected to a single point then no multiplexing is needed. If, on the other hand, the point is connected to k $(k > 1)$ ports then a k-to-1 multiplexer, will be needed. Such a multiplexer is essentially a set of k switches which operate in a mutually exclusive manner. The interconnect cost for a k-to-1 multiplexer is then taken as k. The third assumption ensures that either $k = 1$ or $k = 3$. If $k = 1$ then interconnect cost is zero and if $k = 3$ then cost is three. This special case, to be referred to as PA3UA1, will help us to use the complexity results of GN3D.

Given an instance of PA3UA1 we make the following construction. A graph is constructed from the set of transfers for the various control steps. If points p_1 and p_2 access the memory in the same control step then they must access the memory through distinct ports. In the graph an edge is introduced between the vertices corresponding to p_1 and p_2. If this graph is 3-colourable then a feasible assignment of the the points to the ports can be directly obtained. Otherwise, it will be necessary to connect some of the points to more than one port, to satisfy the memory accesses. Since we are working on an instance of PA3UA1, we shall connect such a point to all the three ports of the memory. All the conflicts for such a point can always be resolved, and the vertex corresponding to this point may be deleted from the conflict graph. To minimize the interconnect cost the number of such points should be minimized. Clearly this corresponds to the general node deletion problem to achieve three colourability (GN3D).

4.2 Complexity of PA3UA1

It will be shown in this section that PA3UA1 is just as hard as GN3D. Thus an approximation for PA3UA1 whose relative error is bounded by a fixed constant is NP-complete. The transformation described previously leads to the following theorem.

Theorem 5 *The relaxed formulation, PA3UA1, of the port assignment problem of three port memories is NP-complete.*

Proof: Given a graph $G(V, E)$ for GN3D, an instance of PA3UA1 will be constructed as follows. Let r_1, r_2 and r_3 be registers packed into the given memory. For each vertex $v_i \in V$ let p_i be a point in the circuit accessing the memory. For each edge $(v_{i_1}, v_{i_2}) \in E$, find a vertex $v_{i_3} \in V$, if such a vertex exists, such that $(v_{i_1}, v_{i_3}) \in E$ and $(v_{i_2}, v_{i_3}) \in E$. If v_{i_3} exists construct the control step
$\mathbf{p_{i_1} \leftarrow r_1}, \quad \mathbf{p_{i_2} \leftarrow r_2}, \quad \mathbf{p_{i_3} \leftarrow r_3},$
otherwise construct the control step
$\mathbf{p_{i_1} \leftarrow r_1}, \quad \mathbf{p_{i_2} \leftarrow r_2}.$

This creates an instance of the port assignment problem with three ports. It is easy to see that the deletion of the vertices whose corresponding points are chosen for connection to all the three ports for conflict resolution will be a feasible solution to GND. This is because the remaining points are connected to single ports and their corresponding vertices may therefore be assigned three distinct colours.

It is easy to see that PA3UA1 is in NP. □

Now, a relative approximate algorithm for PA3UA1 may also be used as an approximate algorithm for GN3D with the same error. Such an approximation for PA3UA1 will be referred to a PA3UA1-R. Therefore from corollary 4 the following corollary follows.

Corollary 6 *An approximation for PA3UA1, PA3UA1-R, whose relative error is bounded by a fixed constant is NP-complete.*

Now the complexity of the original three port problem, viz. PA3U, will be analyzed and it will be shown that it is also as difficult as PA3UA1.

4.3 Complexity of PA3U

We first consider a special case of PA3U, PA3U1 which is as follows. *The accesses to the memory in question are only read accesses, the points that read from the memory do not read from or write to any other place in the circuit.* We note that PA3U1 and PA3UA1 differ only in the way the points may be connected to the ports of the memory. In the case of PA3UA1 a point may be connected to either one or all of the ports. Thus either no multiplexer is needed or a 3-to-1 multiplexer is needed. However, for PA3U1 the point may be connected to one, two or all three of the ports. In this case either no multiplexer is required or a 2-to-1 or a 3-to-1 multiplexer is required at that point. We prove the following theorems for PA3U1. These results directly carry over to PA3U, it being a generalization of PA3U1.

Theorem 7 *PA3U1 is NP-complete.*

Proof: Proved by reducing PA3UA1-R to PA3U1. Let $\mathbf{Y}^\star$ be the cost of the optimal solution to PA3U1 and $\mathbf{X}^\star$, the cost of the optimal solution to PA3UA1. Clearly $\mathbf{Y}^\star \leq \mathbf{X}^\star$. Let an optimal solution to PA3U1 consist of $p_2^\star$ connected to two ports and $p_3^\star$ connected to three ports.

Thus, $\mathbf{Y}^\star = 2p_2^\star + 3p_3^\star \geq 2p^\star$, where $p^\star = p_2^\star + p_3^\star$. Thus, $p^\star \leq \frac{1}{2}\mathbf{Y}^\star$.

The $p^\star$ points of an optimal solution to PA3U1 may be connected to all the three ports, so that this modified solution serves as an approximate solution to PA3UA1. Let $\mathbf{X}$, be the cost of this approximate solution to PA3UA1.
$\mathbf{X} = 3p^\star \leq \frac{3}{2}\mathbf{Y}^\star \leq \frac{3}{2}\mathbf{X}^\star$.

Thus the optimal algorithm for PA3U1 could be used to solve PA3UA1-R, proving that PA3U1 is NP-Hard. It is easy to show that PA3U1 is in NP. □

Corollary 8 *PA3U is NP-complete.*

Proof: NP-hardness of PA3U follows from lemma 7 because PA3U1 is a special case of PA3U. That it is in NP can be shown easily. □

Theorem 9 *An approximation for PA3U1, PA3U1-R, whose relative error is bounded by a fixed constant is NP-complete.*

Proof: Let $\mathbf{Y}$ be the cost obtained by an algorithm for PA3U1-R. Let $\mathbf{Y}^\star$ be the cost of the optimal solution to PA3U1. $\mathbf{Y} \leq \mathbf{kY}^\star$. Let the sub-optimal solution to PA3U1 consist of p_2 connected to two ports and p_3 connected to three ports.
Thus, $\mathbf{Y} = 2p_2 + 3p_3 \geq 2p$, where $p = p_2 + p_3$. Thus, $p \leq \frac{1}{2}\mathbf{Y}$.
In the lines of lemma 7 this solution could also be used as a solution for PA3UA1-R. Let $\mathbf{X}$, be the cost of this solution to PA3UA1-R.
$\mathbf{X} = 3p \leq \frac{3}{2}\mathbf{Y} \leq \frac{3k}{2}\mathbf{Y}^\star \leq \frac{3k}{2}\mathbf{X}^\star$, where $\mathbf{X}^\star$ is the cost of the optimal solution to PA3UA1.
□

Corollary 10 *An approximation for PA3U, PA3U-R, whose relative error is bounded by a fixed constant is NP-complete.*

Proof: Follows from theorem 9, as PA3U1-R is a special case of PA3U-R. □

This completes our treatment of the port assignment problem for triple port memories. We shall use the results derived here in the subsequent sections of this paper. We now go over to register optimization and register-interconnect optimization problems.

5 Register Optimization

The aim of register optimization (RO) is to minimize the number of registers needed in the design [4]. Registers need to be used to store values between control steps. In the context of data path synthesis registers are needed to implement the variables used to describe the behaviour of the target system. In addition to the variables declared by the designer, some variables may be used at the time of generating intermediate code. All variables in the final implementation need to be mapped onto registers. It may be possible to map some of these registers onto on-chip memories. Registers and variables will be used in an interchangeable manner.

A variable is live from the time when it is first defined till the time that value is last used. A variable may become live several times during the execution of the program. Two variables that are never live at the same time may be merged and placed on the same register without affecting the logical input/output behaviour of the program. It is necessary to determine the life times of each variable in a program. This is called live variable analysis [7]. Once the life times of the variables are known it is necessary to represent their sharability and perform register minimization.

5.1 RO for Straight Line Code – A Solved Problem

We call a patch of code that contains neither branching instructions nor targets of branching instructions, a *straight line code*. Such code may be represented by a single DAG. The register optimization for this case will be referred to as SRO. In practice some behaviours encountered are operation intensive and do not contain any decision making branches or looping constructs at all. Sometimes loops with fixed number of iterations may be unrolled to remove the iteration. The intermediate code of such behaviours take a particularly simple form, consisting of only arithmetic or logical operations. For such kind of code it turns out that the complement of the vertex compatibility graph of the variables in the DAG is an interval graph. The complemented vertex compatibility graph (CVCG) in this case being an interval graph may be optimally coloured, using the *left edge algorithm* [4], in polynomial time, V being the set of vertices in the CVCG.

5.2 General RO

In general, however, the sharability will not be as simple as that for SRO and may be represented by a graph where there is a vertex for each variable. Two vertices are joined by an edge if the lifetimes of the corresponding variables are disjoint. This graph will be called the vertex compatibility graph (VCG). The problem of register minimization may now be mapped to the clique partitioning problem (CP), which is to find the minimum number of disjoint cliques that cover a graph. Each clique in the VCG corresponds to a set of variables to be mapped on a single register. This general case of RO will be called GRO. The following result can be easily proved.

Theorem 11 *GRO is NP-complete.*

A comparative study of the complexity of the Traveling Salesperson Problem, Clique, Colouring and Bin Packing has been made in [8]. It has been shown in [9] that no polynomial time approximate algorithm is currently known for graph colouring for which the bound on the relative error is even close to ∞. A very recent result in [10] states that, for the colouring problem there is constant $\epsilon > 0$ such that no polynomial time approximation algorithm can achieve a ratio of n^ϵ (to the optimal) unless P=NP. This leads to the following result.

Theorem 12 *The relative approximation of GRO, GRO-R, is NP-hard.*

In the next section we consider the problem of simultaneously optimizing the register and the multiplexer cost. We call this register-interconnect optimization (RIO). RIO based on flexible variable binding could be applied to *intermediate variables*, by breaking up a single contiguous life time. This would not lead to a reduction in the number of variables, but could help to reduce interconnection overhead.

6 Register-Interconnect Optimization

Pure RO permits the minimization of registers. However, it has been seen in several design examples that pure RO leads to inferior designs, with excessively high interconnect cost. The interconnect cost may be estimated by counting the total number of multiplexer channels needed at the inputs of the hardware elements used in the circuit. RO performed along with interconnect optimization is called register-interconnect optimization (RIO). The formulation of RIO is similar to RO and is briefly presented below.

The register sharability information is identical to that for RO. In addition a description of the logical netlist is also presented to evaluate the effect of merging two registers on the interconnect cost. For a particular merger the change in multiplexer cost may be computed. The net list needs to be updated at each step to indicate the merger of the two registers. The net effect of the register mergers at a certain time may be computed from the updated netlist of the design. The RIO problem may be formulated as that of finding a set of mergers of registers such that a weighted sum of the register and the multiplexer cost is minimized. Thus the objective function to be minimized may be taken as

$$C = w_1 n_r C_R + w_2 n_m C_M, \tag{1}$$

where C_R is the register cost and C_M is the unit multiplexer switch cost; n_r and n_m are the number of registers and the multiplexer channels, respectively. The cost of a register is proportional to the number of bits in the register. The cost of a multiplexer is proportional to the number of lines being multiplexed and the width of its output. If w_1 and w_2 are both taken as 1, in equation 1 then the total cost for the registers and the multiplexer channels will be minimized.

If we consider the RIO problem for a special class of circuits where n_m in equation 1 will be zero, we immediately get a reduction of RIO to RO. This directly leads to the following result.

Theorem 13 *RIO is NP-complete.*

As for RO it is possible to distinguish between general RIO (GRIO) and RIO for straight line code (SRIO). RIO being NP-complete GRIO is also NP-complete. It will be shown in section 6.1 that SRIO is also NP-complete. Thus, even though SRO is solvable optimally in polynomial time, using the *left edge* algorithm, SRIO is not. In fact, even approximation of SRIO is hard.

6.0.1 Reduction of PA3U to SRIO

This section presents a reduction of the port assignment problem for three ports, PA3U, to RIO for straight line code, SRIO, and, therefore, to RIO also. The PA3U case consists of a set of transfers between the memory and the points accessing the memory in the various control steps. Each such transfer must take place through a port of the memory. Since only three ports are permitted, in any control step no more than three simultaneous accesses to the memory may be allowed. Since all the ports are uniform, it is permissible to map a transfer on any of the three ports. The mapping is done in two phases. First a mapping is made to SRO. This mapping does not ensure optimality. It is only a prelude to a mapping to SRIO which does ensure obtaining an optimal solution.

The mapping is explained with the help of example 2. In this example each access to the memory is denoted as a_{it}, for the i-th access in time step t. While constructing an instance of SRO for the PA problem each such access plays the role of a register. Thus each a_{it} is mapped to a register r_l, such that the mapping $< i, t > \rightarrow l$ is unique. It is evident from the construction that the lifetime of the register r_l is the time step t in which the access a_{it} takes place. The register lifetimes form and interval graph, as expected. This is how the mapping from an instance of triple port memory PA to SRO is achieved.

The total number of registers live in any time step will not exceed three, which is the maximum number of accesses to the memory in any time step. It is, therefore, ensured that there exists an non-empty set of solutions to the SRO problem requiring not more that three registers. These are the solutions that we shall consider. Such a solution will always be obtained by running the left edge algorithm. Each register of the solution to the SRO problem instance actually corresponds to a grouping of memory accesses in different time steps. This can, indeed, be considered an assignment of this transfers to one of the ports of the memory. Since the solutions considered contain no more than three ports, each solution may be used as a feasible solution to the PA problem.

Example 2 The creation of the problem instance for SRO and SRIO from a triple port memory PA problem of example 1 is now explained. Shown below are the transfers, the corresponding accesses and the variables for the SRO and SRIO instance. The variables a, b, c and d are packed into a memory for which the PA needs to be done. The points $l_a, l_b, \ldots$ are as in example 1. (refer to figure 1).

	Transfers		Accesses	Registers
1.	$a = b + c;$		a_{11}, a_{21}, a_{31}	r_1, r_2, r_3
2.	$q = c + d,$	$b = p - q;$	a_{12}, a_{22}, a_{32}	r_4, r_5, r_6
3.	$d = a + r,$	$c = p - r;$	a_{13}, a_{23}, a_{33}	r_7, r_8, r_9
4.	$a = p + c,$	$b = q - r;$	a_{14}, a_{24}, a_{34}	r_{10}, r_{11}, r_{12}

The transfers for the SRIO instance which are needed to construct the netlist are shown below.

1.	$r_1 \leftarrow o_a$	$l_a \leftarrow r_2$	$r_a \leftarrow r_3$
2.	$l_a \leftarrow r_4$	$r_a \leftarrow r_5$	$r_6 \leftarrow o_s$
3.	$r_7 \leftarrow o_a$	$l_a \leftarrow r_8$	$r_9 \leftarrow o_s$
4.	$r_{10} \leftarrow o_a$	$r_{11} \leftarrow o_s$	$r_{12} \leftarrow o_s$

□

The above mapping to SRO, does not attempt to restrict the multiplexer usage and so the interconnect cost may be expected to be high. An optimal assignment may be obtained by mapping the problem to SRIO. The register lifetimes are same as for SRO. Only the interconnect information needs to be incorporated. The method of extraction of interconnection information from the transfers has already been illustrated in example 2, is as follows. Each point accessing the memory now plays the role of a point in a circuit in the SRIO instance. For a transfer between a point p and the memory in access a_{it}, in the PA example, a transfer between register r_l and point p is created in the SRIO instance.

We now complete the mapping to SRIO. In order to ensure that the solution will not require more than three ports, we fix the weights in equation 1 suitably. Suppose that the total number of circuit points accessing the memory is m. In the worst case each point will be connected to all the ports and the number of multiplexer lines needed will be $3m$. This is a upper bound on the number of multiplexer lines needed for the port assignment. In equation 1, let w_1 be set to $(3m+2)$ and w_2 be set to 1. Any feasible solution will have cost $c \leq 12m+6$. It is assumed that C_R and C_M have each been set to 1.

6.1 Complexity of RIO

It was shown in section 6 that RIO is NP-complete. It will now be shown that an approximation for RIO, RIO-R, whose relative error is bounded by a fixed constant is NP-complete. This will be proved through a sequence of reductions. First PA3UA1-R is reduced to SRIO-R. SRIO-R is directly reducible to RIO-R and SRIO. This way SRIO-R and RIO-R will be proved to be NP-hard.

Consider an approximation for SRIO, SRIO-R, whose relative error is bounded by a constant. It has been shown that PA3UA1-R is NP-complete. It will be shown that the approximation PA3UA1-R is reducible to the approximation SRIO-R.

Theorem 14 *SRIO-R is NP-complete.*

Proof: Given an instance of PA3UA1-R, which is essentially a PA problem for triple port memory, consider the following. Let $\mathbf{A}^\star$ be an optimal algorithm for PA3UA1. Let $\mathbf{X}^\star$ be the cost of the solution obtained by $\mathbf{A}^\star$, i.e. the multiplexer lines needed as a result of connecting some of the circuit points to all the three ports.

Let $\mathbf{B}^\star$ be an optimal algorithm for SRIO. Let $\mathbf{B}$ be an algorithm for SRIO-R. Let $\mathbf{Y}^\star$ be the effective cost of the solution obtained by $\mathbf{B}^\star$, i.e. total number of multiplexer lines used for the port assignment. Let $\mathbf{Y}$ be the cost of the solution obtained by $\mathbf{B}$. $\mathbf{Y} \leq k_2\mathbf{Y}^\star$.

$\mathbf{Y}^\star \leq \mathbf{X}^\star$, since the mapping of section 6.0.1 ensures that $\mathbf{B}^\star$ yields an optimal solution to PA3U1 and because PA3UA1 is a sub-optimal formulation for PA3U1.

Suppose that the solution obtained be $\mathbf{B}$ requires p_2 two-to-one multiplexers and p_3 three-to-one multiplexers.

Then, $\mathbf{Y} = 2p_2 + 3p_3 \Rightarrow p_2 + p_3 \leq \frac{1}{2}\mathbf{Y} \Rightarrow p_2 + p_3 \leq \frac{k_2}{2}\mathbf{Y}^\star \leq \frac{k_2}{2}\mathbf{X}^\star$

The sum $(p_2 + p_3)$ is the total number of points which are connected to more than one point by $\mathbf{B}$. A sub-optimal solution to PA3UA1 may be obtained by connecting these points to all the three ports of the memory. The cost of this solution would be $3(p_1 + p_2) = \mathbf{X}$.

Thus, $\mathbf{X} = 3(p_1 + p_2) \leq \frac{3k_2}{2}\mathbf{X}^\star \leq k_1^{'}\mathbf{X}^\star$

But the error in the sub-optimal solution obtained by the above method for PA3UA is bounded by a fixed constant. The above method may therefore be used as an algorithm for PA3UA1-R. Since PA3UA1-R is NP-complete, it follows that SRIO-R is NP-Hard.

It may be easily proved that SRIO-R is in NP. □

The following three corollaries follow from theorem 14.

Corollary 15 *SRIO is NP-complete.*

Corollary 16 *An approximation for RIO, RIO-R, for which the relative error in the solution is bounded by a fixed constant is NP-complete.*

From corollary 16 also it follows that RIO is NP-hard.

7 The Problem of Forming Functional Units

We have explained earlier that the design of the data paths starts with the scheduling of the DAG's and is followed by allocation and binding. We consider a practical situation where the design has to be performed subject to a constraint that a specified number of functional units has to be used. This constraint leads to the requirement that no more than this number of operations should be scheduled in any time step. This a small departure from a prevalent approach to the scheduling problem where the sum total of the cost of hardware operators is to be minimized without consideration for the number of units where these operators are housed.

With both the approaches the objective is to minimize the cost of hardware operators. Scheduling algorithms for the prevailing approach seek to minimize the maximum number of operations of each kind that will be performed in any time step. The actual formation of the FUs is usually done by binding the operations to the functional units (FU) during the allocation and binding step. This process is illustrated and clarified in example 3.

Example 3 Consider the schedule of operations in example 1. The scheduling algorithm would report that a maximum of one addition and one subtraction are performed in each time step. Thus a minimum of one adder and one subtracter needs to be allocated. The scheduling has been done so that a maximum of two operations are scheduled per time step, so that two FUs would suffice to actually accommodate each operation in each time step. All the additions could be mapped onto one of the FUs and all the subtractions could be mapped onto the other. The resulting FUs would be as shown in figure 1. □

Example 4 Now consider the following transfers.

1. $q = c + d, \quad b = p - q;$
2. $d = a + r, \quad c = p \,\&\, r;$
3. $a = p \,\&\, c, \quad b = q - r;$

As in the case of example 3 a maximum of two operations are scheduled per time step and so two FUs would suffice to actually accommodate the operations in each time step. The scheduling algorithm would report a maximum of one adder, one subtracter and one *word-and gate*. However, an inspection will reveal that an assignment of these operations to the FUs such that at most one FU implements one operation is not possible. At least one of the operations would have to be assigned to two FUs in order to satisfy the specified schedule. Thus we see that the allocation of one operation of each kind which is implied by the schedule, cannot be satisfied in this case. A set of FUs resulting from such an assignment would be $< +, - >$, and $< +, \& >$. □

Example 4 actually reveals a severe difficulty faced while performing scheduling. While scheduling it is desirable to find a schedule which is realizable using a set of FUs of minimum cost. The usual method of estimating the cost of the operator set that would be required to actually realize the operations in the schedule is to sum up the costs corresponding to the maximum number of operations of each kind. As it has been indicated in example 4 this method is inadequate, as in final assignment of operations to the FUs some more operations of each kind could be required, thus violating the allocation implied by the schedule.

The decision problem of forming FUs from the allocation implied by the schedule is formally defined as follows. We are given a schedule of operations, each of a single time step. The schedule consists of $p, p > 0$, types of single time cycle operations. At most $n, n \geq 3$, operations are scheduled in each time step. The maximum number of operations of type i in any time step of the schedule is $m_i, m_i \leq n$. The problem is to determine whether an assignment exists of the operations in each time step to n FUs, such that no two operations in the same time step are mapped to the same FU and no more than m_i of the FUs implement operation of type i. We shall refer to this problem as FUFD.

We would like to determine the complexity of the above problem. To do this we consider the problem special case when there is at most one operation of a particular kind scheduled in any time step, i.e. $m_i = 1$ and a fixed number k of FUs are to be used. This special case will be referred to as FUFD1(k), for which we have the following result.

Theorem 17 *For a fixed integer k, FUFD1(k) is NP-hard.*

Proof: Given a graph $G(V, E)$ for GND, an instance of FUFD1(k) will be constructed as follows. For each vertex $v_i \in V$ let O_i be a type of operation in the schedule. For each clique of size l, $l \leq k$, construct a time step where the operations scheduled are as follows:

$\mathbf{o_{i_1}}, \quad \mathbf{o_{i_2}}, \ldots \quad \mathbf{o_{i_l}},$

where o_j is an operation of type O_j.

This creates an instance of FUFD1(k) in polynomial time when k is fixed. The graph will be k colourable if and only if an assignment of the operations to the k FUs exist such that no operation is implemented by more than a single FU $\square$

We need not be restricted to designing FUs where the usage of individual operations does not violate the initial allocation. Our main concern is to get FUs of minimum cost so that the schedule that has been obtained can be realized. However, as the decision problem of designing the FUs with a fixed number of FUs is hard, as shown in theorem 17, the aforesaid minimization problem is also hard.

Theorem 17 establishes the NP-hardness for this problem only when three or more FUs are to be used. If only one FU is to be used then the problem is trivial. We shall now examine the problem when only two FUs are to be used.

When only two FUs are to be used it follows that at most two operations may be scheduled in any time step. Suppose that a maximum of one operation of any kind is present in a time step. It is now possible to find out in polynomial time whether the required assignment of operations will exist or not. This is because the problem of finding the bipartite partition of a graph, if one exists, can be done in polynomial time. However, for our purpose we need the actual assignment of operations to the FUs so that the cost of the units is minimized. We now show that this problem is NP-hard.

Theorem 18 *The problem of determining the actual assignment of single cycle operations to FUs when exactly two FUs are to be used, so as to minimize the cost of the FUs is NP-hard.*

Proof: Consider the special case when exactly one operation of each type is present and an operator for each operation has the same cost. Construct a conflict graph G as explained the proof of theorem 17. It would now be necessary to two colour the graph with just two colours. If the colouring succeeds then an assignment exists that does not violate the default allocation. However, if the colouring fails, then it becomes necessary which of these operations should be assigned to both the FUs. In order to minimize the cost of the formation of the FUs we would like to assign as few of these operations, as possible, to both the FUs. This corresponds exactly to performing minimum node deletion on the graph G. The operations corresponding to the nodes that have been deleted would have to be assigned to both the FUs. It is well known that minimum node deletion is NP-complete and this proves the theorem. □

Theorem 19 *The problem of determining the assignment of operations to a fixed number k, $k > 1$ of FUs, so as to minimize the cost of the FUs is NP-hard.*

Proof: It is easy to show that this problem is in NP. The NP-hardness of this problem follows from theorem 17 and theorem 18. □

8 Conclusions

In this paper we have examined the complexity of several synthesis tasks that are encountered during allocation and binding. The problems that we have considered are the port assignment problem for triple port memories, the register-interconnect optimization problem and the problem of formation of functional units. We have shown that all these problems in general are NP-complete.

The port assignment problem has recently acquired significance because triple port memories are now being used on-chip. Moreover, we have used the complexity results of PA to derive the complexity results for RIO.

We have also examined the complexity of the problem of minimizing the cost of functional units, when it is necessary to make a design where the total number of FUs is specified before hand. We have shown that this problem is also NP-hard.

We have examined several versions of the RIO problem. The simplest case that we have considered is RO for straight line code which is solvable in polynomial time. The next more general case that we have considered is RIO for straight-line code, which we have shown to be NP-hard. All the results of SRIO carry over to RIO because SRIO is a special case of RIO. Some results on connectivity binding were already available [2]. Our results have been derived using a different approach which produces newer insights into the hardness of the interconnect optimization problem. The most significant contribution of our work on the complexity of allocation and binding are the hardness results of the relative approximation of several sub-problems in this area. We have shown that the constant bounded relative approximation of PA for triple port memories and SRIO are both NP-hard. These hardness results suggest that the use of deterministic approaches to very global formulation of allocation and binding may not prove fruitful, in general. Non-deterministic optimization such as simulated annealing [11] and genetic algorithms [12] are increasingly being used in DPS.

References

[1] C. J. Tseng and D. P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Transactions on Computer Aided Design*, vol. CAD-5, July 1986.

[2] B. M. Pangrle, "On the complexity of connectivity binding," *IEEE Transactions on Computer Aided Design*, vol. 10, pp. 1460–1465, November 1991.

[3] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Port assignment for dual and triple port memories using a genetic approach," in *Proceedings of The Third Asia Pacific Conference on Hardware Description Languages (APCHDL) 96*, pp. 60–64, 1996.

[4] F. J. Kurdahi and A. C. Parker, "Real: A program for register allocation," *Proceedings of the 24th Design Automation Conference*, 1987.

[5] E. Horowitz and S. Sahni, *Computer Algorithms*. Galgotia Press, New Delhi, INDIA, 1988.

[6] T. C. Wilson, D. K. Banerjee, A. Basu, J. C. Majithia, and A. K. Majumdar, "Port assignment in multiport memories for interconnection minimization in data path synthesis," in *Proceedings of IFIP Working Conference on Logic and Architecture Synthesis, Paris*, May 1990.

[7] A. V. Aho, R. Sethi, and J. D. Ullman, *COMPILERS Principles, Techniques and Tools*. Addison-Wesley Publishing Company, June 1987.

[8] M. W. Krentel, "The complexity of optimization problems," *Journal of Computer and System Sciences*, vol. 36, pp. 490–509, 1988.

[9] D. S. Johnson, "Worst case behaviour of graph colouring algoorithms," *Prooceedings of 5th South Eastern Conference on Combinatorics, Graph Theory & Computing*, 1974.

[10] C. Lund and Y. Mihalis, "On the hardness of approximate minimization problems," in *Proceedings of the* 25^{th} *Annual ACM Symposium of the Theory of Computing*, 1993.

[11] A. A. Duncan and D. C. Hendry, "Area efficient dsp synthesis," in *Proceedings of the 1995 European Design Automation Conference*, pp. 130–135, September 1995.

[12] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Allocation and binding for data path synthesis using a genetic approach," in *Proceedings of VLSI Design '96*, pp. 95–99, 1996.