Constrained Register Allocation in Bus Architectures

Elof Frank
German National Research Center
for Computer Science (GMD)
D - 53754 St. Augustin
Germany
elof.frank@cartan.gmd.de

Salil Raje and *Majid Sarrafzadeh*
Northwestern University
Electrical and Computer Science
Evanston, IL 60208
USA
{salil, majid}@eecs.nwu.edu

Abstract

Partitioned memory with bus interconnect architecture in its most general form consists of several functional units with associated memory accessible to the functional unit via local interconnect and global buses to communicate data values across from one functional unit to another. As can be expected, the time at which certain values are communicated affect the size of the local memories and the number of buses that are needed. We address the problem of scheduling communications in a bus architecture under memory constraints. We present here a network flow **formulation** *for the problem and obtain an* **exact** *algorithm to schedule the communications, such that the constraint on the number of registers in each functional unit is satisfied. As an increasing number of architectures use multiple memories in addition to (or instead of) one central RAM, this work is especially interesting. Several authors have already studied this problem in related architectures, yet all use heuristic approaches to schedule the communications. Our technique is the first exact solution to the problem. Also, our graph theoretic formulation provides a clearer insight into the problem.*

1 Introduction

A common approach to high-level synthesis involves a data-flow graph scheduling and functional unit allocation and then proceeds to interconnect and storage allocation. In bus based architectures, there is a direct correlation between the time for transporting values over the buses and the number of buses and registers needed.

32nd ACM/IEEE Design Automation Conference ®

If only one main memory (RAM, register file) is available for all the variables, the time of communication is fixed to the execution time of the operation using this value. In the fetch phase of its communication time, the value will be loaded onto a bus and will then be directly fed into the port of the appropriate functional unit. Therefore, the number of buses needed is given by the maximum number of distinct values consumed by all functional units at each cstep. The number of registers is given by the maximum density of all lifetimes of the variables. Several authors use this target architecture to validate their scheduling and estimate interconnect and storage cost ([10, 8, 2, 6]).

On the contrary, having local memory (registers, register file, distributed RAM's) in each functional unit allows for the flexibility of scheduling these communications onto specific time steps. A simple architecture enabling this freedom is given in Figure 1. In this architecture a functional unit consists of an ALU and a register file. Variables are transported over a global bus, if the source FU and the target FU are different. There is a local interconnect structure between the functional unit and its associated register file. A control step consists of two micro operations (fetch, execute). A value which is produced at a time step t will be stored in the local register file of its source functional unit. It is then available for transportation on the bus in the following fetch phase at the earliest and has to be communicated at the latest when it is needed at the target FU. We refer to these two time steps as the *fat (or first available time)* and *lat (last available time)* times of a communication. Note that the schedule of the data-flow graph is already known (this is an input to our problem) and the *fat* and the *lat* for each of the data values are known a priori.

This mobility range (*fat*, *lat*) for communications gives us the flexibility to choose specific time steps for each communication. This influences register and bus cost. We present a resource constrained approach for

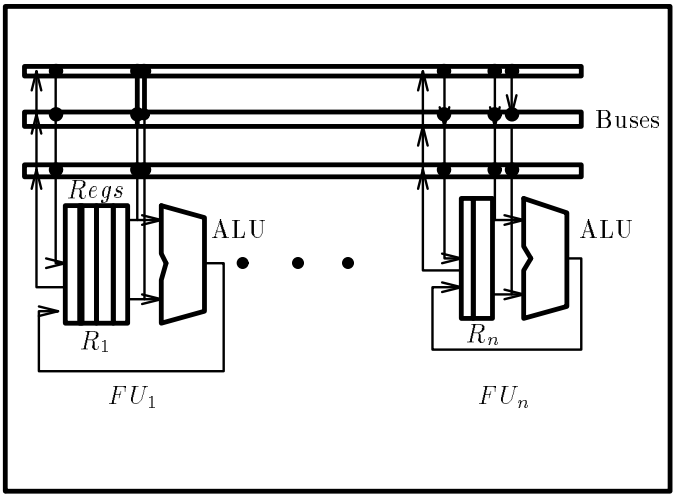

Figure 1: Architecture of the data path with local register files

determining these times. In this paper we only regard constraints on the size of the register files, but equivalent constraints can also be imposed on the number of buses.

1.1 Related Research

Several authors have presented methodologies and heuristics to solve the problem of scheduling communications (known also as data routing ([7])). In the Cathedral-II system [1] the synthesis process is directed towards a similar architecture, yet they do not exploit the freedom of scheduling communications. Every value is stored in the source functional unit and communicated on a bus when it is needed, i.e. as late as possible. The buses are allocated by merging point-to-point wires, which do not conflict in time.

Lanneer et al. [7] present a general data routing paradigm. In their architecture a value not only has the possibility to be scheduled at different time steps but also on different global routes. A global route is an interconnection between storage devices, but in their architecture it does not necessarily have to be a bus. They do not give an analysis on the complexity of the problem even for simpler architectures. Rather, they present an efficient heuristic to the problem based on a force-directed scheduling approach [9].

In [3], the problem of scheduling communications is regarded in a related architecture too, yet the authors solve the problem heuristically as well. Our approach can be adopted to their target architecture.

Ewering [4] schedules communications on a partitioned bus architecture. Partitioned bus architectures have the advantage that different values can be scheduled onto one bus track at the same time, as long as their global routes do not overlap. This is achieved by introducing switches on the buses and thereby being able to physically partition each bus by desire. The approach taken by Ewering is to solve the scheduling of communications and the assignment of communications to specific bus tracks at the same time. This is achieved by enumerating all possible routes for a communication and solving a maximum independent set problem to decide which communications do not conflict each other. Their algorithm works by performing an exhaustive search and they do not present an analysis for the runtime of their approach.

In this paper we solve the problem of scheduling communication graphs, given constraints on the number of registers in each functional unit. Our paper emphasizes on two aspects. At first, we model our problem as a new and interesting network flow problem, which we call a commodity constrained network flow problem. We solve the problem, by performing an intelligent exhaustive search on this network. The runtime can be upper bounded by an exponentiation in the number of time steps, compared to a simple exhaustive search, which has an exponentiation in the number of communications. As the number of communications in general is large, compared to the number of time steps, this leads to a sophisticated improvement.

The rest of our paper is organized as follows: We will start with giving a formal definition to the problem of register constrained scheduling of communications on our architecture. Although our methodology can be applied to several settings (bus constraints, memory port constraints, partitioned bus architectures), we will only present the approach taken for single target communications (each value is only consumed by one functional unit) without bus or port constraints for purposes of simplicity. In section 3 we reduce the problem of scheduling communications to a special maxflow problem (which we call a commodity constrained network flow) and present a modified augmenting path method for the Ford Fulkerson Method [5], which solves this problem **exactly**. The augmenting path method performs an exhaustive search, which is bounded by $O((m/n)^T)$, where m is the number of communications, n is the number of functional units and T is the number of time steps. We conclude in section 5.

2 Definitions and Problem Formulation

The input to our problem is a scheduled data-flow graph with each of the operations in the data-flow graph bound to a specific functional unit. The task to be performed is to schedule the communications of the data values at specific times such that the local register file sizes satisfy the constraints imposed on them. A *Communication Graph* is a directed graph $G = (V, E)$. V is the set of modules (functional units) as given by the allocation and operation binding. An edge $e = (v, w) \in E$ has one source v and one target w. Each edge e is labeled with a pair (fat, lat). e represents a value, which is produced in functional unit v at time fat and is consumed at functional units w at time lat. In essence, a communication graph is a folded data-flow graph , where the operations of the data-flow are merged to one node, representing the functional unit executing these operations.

Figure 2.1 shows an instance of a scheduled data-flow graph and a resulting communication graph, given by a module allocation using one adder **ADD** and one subtracter **SUB**. Two possible schedules are given in Figure 2.2, the first resulting in an allocation of two buses and three registers, the second resulting in two buses and four registers. The only difference is in the scheduling time for the value c_2, which, in the second case is scheduled as late as possible, resulting in a register increase in the functional unit **SUB** at time step 3.

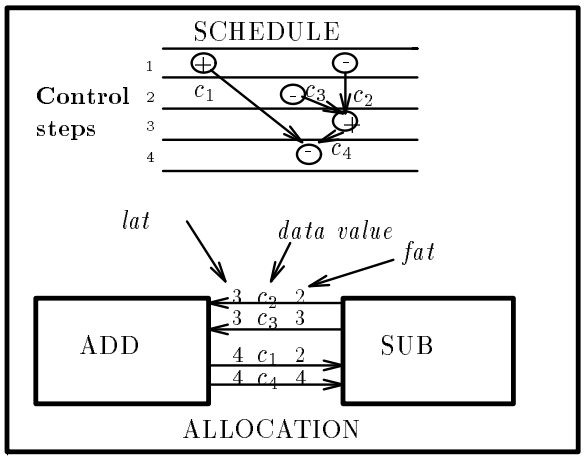

Figure 2.1: Input to our problem: a) Scheduled data-flow graph with five operations b) Communication graph with allocation of two functional units.

Definition 2.1 *Given communications* $e_1, \ldots, e_n$ *with scheduling times* $\tau_1, \ldots, \tau_n$.

An edge e_i *is* **alive** *at time* t *in functional unit* v_j, *iff* $fat_i \leq t \leq \tau_i$ *and* $e_i = (v_j, v_l)$ *or* $\tau_i \leq t \leq lat_i$ *and* $e_i = (v_l, v_j)$.

Define the live set $L_{j,t}$ *of functional unit* fu_j *at time* t *as*

$$L_{j,t} = \{e_i | e_i \text{ is alive in } v_j \text{ at time } t\}. \quad (1)$$

We define the **density** $D_{j,t}$ *of* $v_j \in V$ *at time* t *as*

$$D_{j,t} = |L_{j,t}| \quad (2)$$

$D_{j,t}$ *is equivalent to the number of registers needed in* v_j *at time* t.

$$D_j = \max_{1 \leq t \leq T} D_{j,t} \quad (3)$$

defines the size of the memory (e.g. the number of storage locations) needed in functional unit v_j *for the given schedule.*

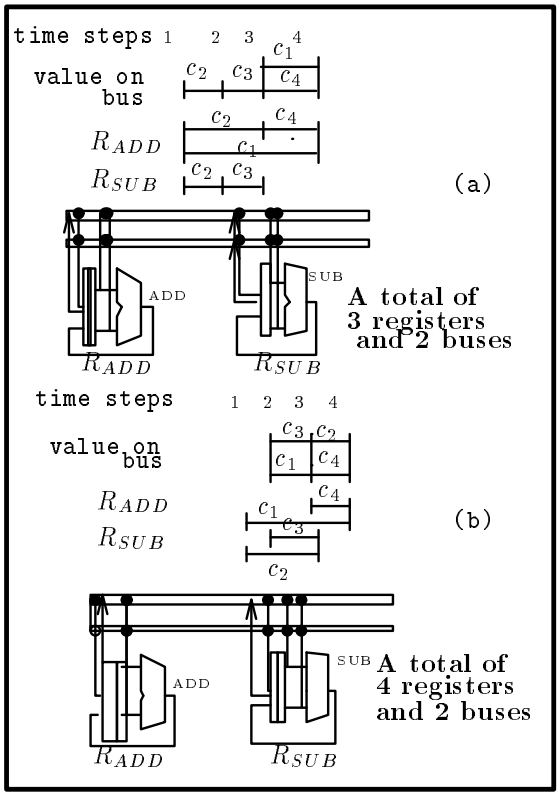

Figure 2.2: Two different schedules for the communication graph resulting in 2 buses and 3 registers in the first case and in 2 buses and 4 registers in the second case. The input to the problem is the communication graph in Figure 2.1.

Problem Formulation:

Given a communication graph $G = (V, E)$ and a set of integers $R_1, \ldots, R_{|V|}$, find a schedule for the communications of E, i.e. for each edge $e_i = (v_s, v_t)$ an integer τ_i, such that $D_j \leq R_j$ for $1 \leq j \leq |V|$.

Example: In the example of Figure 2.2a) the scheduling times are $\tau_1 = 4$, $\tau_2 = 2$, $\tau_3 = 3$, and $\tau_4 = 4$. Edge c_2 is alive in SUB at time 2 and alive in ADD at 2 and 3. The live set $L_{ADD,4} = \{c_1, c_4\}$, $D_{ADD,4} = 2$ and $D_{ADD} = 2$ as well.

3 Commodity Constrained Network Flow

We transform the problem of scheduling communications to a special case of a network flow problem, which we call a *Commodity Constrained Network Flow Problem*. The traditional network flow problems do not suffice our purpose since in these there is no distinction between edges and commodities of different types could flow along any edge as long as they satisfy the capacity constraints. In a commodity constrained network flow problem not every commodity may flow over every edge. There are two types of edges, commodity constrained edges with label e (CC_e-edges) and commodity unconstrained edges (CU-edges). CU-edges can take any type of commodity, whereas CC_e-edges can only take a commodity with label e. In our application the capacity of CC_e-edges is fixed to 1.

3.1 Constructing the Network

The construction of the network flow graph is as follows:

For each memory j and each time step $t = 1, \ldots, T+1$ there is a node $r_{j,t}$ and commodity constrained edges $e_{j,t}$ between $r_{j,t}$ and $r_{j,t+1}$ for $t = 1, \ldots, T$. These edges $e_{j,t}$ have capacity R_j and intuitively the flow $f_{j,t}$ over edge $e_{j,t}$ is equivalent to the value of $D_{j,t}$. By satisfying the capacity constraint $f_{j,t} \leq c_{j,t}$ for all j and all t, we automatically satisfy $D_j \leq R_j$ for all j.

The network has one source node s and one target node t. For each communication edge e with source v_i and target v_j, and earliest and latest times of communication (fat, lat) construct a set of nodes $v_{e,i,t}$ and $v_{e,j,t}$ for $fat \leq t \leq lat + 1$. Furthermore construct a set of CC_e-edges $(v_{e,i,t}, r_{i,t})$ and $(r_{i,t}, v_{e,i,t+1})$ as well as $(v_{e,j,t}, r_{j,t})$ and $(r_{i,t}, v_{e,i,t+1})$ for $fat \leq t \leq lat + 1$. Finally three more CC_e-edges are constructed from s to $v_{e,i,fat}$, from $v_{e,i,lat+1}$ to t and from $v_{e,j,lat+1}$ to t.

For each possible communication time $t \in [fat, lat]$ there exists a bus edge $(v_{e,i,t+1}, v_{e,j,t})$. A bus edge from $v_{e,i,t+1}$ to $v_{e,j,t}$ will let the flow switch from the source register file to the target register file, therefore a flow on one of these edges corresponds to scheduling the value of e at time t. Note that a flow on one of these edges will result in a flow on the register edge $(r_{i,t}, r_{i,t+1})$ as well as on the edge $(r_{j,t}, r_{j,t+1})$. Therefore two registers are allocated for this value at this time step, as our architecture imposes. When regarding constraints on the buses as well, these bus edges will be replaced by a more complex edge structure.

Figure 3.1 shows a snapshot of the commodity constrained network flow graph constructed for the communication edge c_1 in Figure 2.1. All edges with label c_1 are CC_{c_1}-edges.

For those edges with only one possible scheduling time, i.e. $fat == lat$ the scheduling time is fixed to $\tau_e = lat$. These communications are only alive in the source functional unit for one unit of time and it is sufficient to decrease the capacity of edge $r_{i,fat}$ by one, where v_i is the source functional unit of this communication edge.

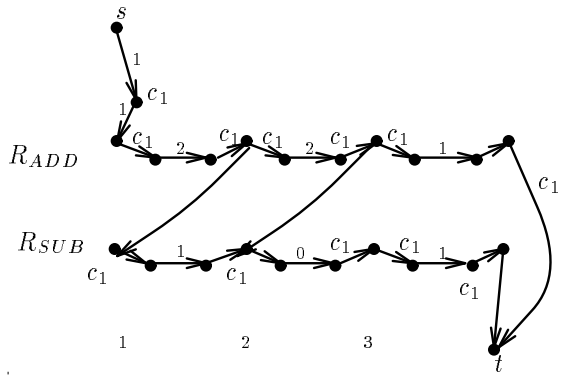

Figure 3.1: There are three possible paths for the one unit flow of this value, corresponding to possible scheduling times 1,2 or 3. Note, that if c_1 is scheduled at *lat*, no register will be allocated in the target functional unit, whereas in the other two cases two registers each are needed at the scheduling times

3.2 Finding Augmenting Paths

Definition 3.1 *A flow for a commodity e is a path from the source s to the target t, which only uses CU-edges and CC_e-edges.*

It is easy to see, that such a flow can use exactly one of the bus edges $(v_{e,i,t+1}, v_{e,j,t})$ for $fat \leq t \leq lat - 1$ or it uses the edge $(v_{e,i,lat+1}, t)$. In the first case this flow is equivalent to scheduling e at time t, after which it

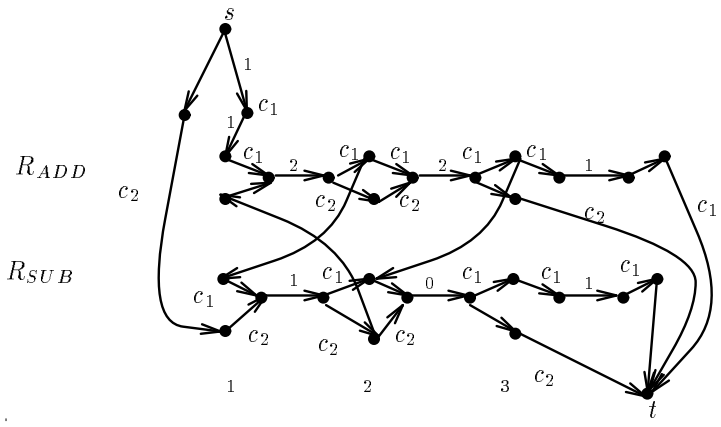

Figure 3.2: The network for the example in Figure 2.1. with resource constraints of 2 for the adder and 1 for the subtracter.

needs to be stored in the memory of functional unit v_j. In the latter case this flow is equivalent to scheduling e at *lat* in which case it does not have to be stored in the memory of v_j (it will be directly fed into the port of the ALU).

An algorithm for this commodity constrained network flow problem uses a modified version of the Ford Fulkerson method, by successively finding an augmenting path for each commodity e.

The basic outline of the algorithm is the following. We schedule all commodities sequentially, but in any order. In the simplest case, scheduling a commodity with label e necessitates finding a path in our network, which only uses commodity constrained edges of label e and commodity unconstrained edges. In general, when looking for an augmenting path for a commodity e, we might find a saturated commodity unconstrained edge CU, because some other commodities already flow through this edge, and the remaining capacity is zero (the register file is full). We start a rescheduling search here. For any commodity e', which flows over the edge CU $= (a, a')$, we try and reschedule the commodity e', by finding a path, which flows from a to a', using only commodity unconstrained edges or commodity constrained edges of label e'. In the pathfinding algorithm there are four cases to be regarded. For each commodity e the search is invoked by finding a path for a commodity e. As we go along, the residual network is updated dynamically. This dynamic updating allows us, to backtrack along paths, if the search does not lead to a success. If the edge is a CC_e-edge, then the path can be continued along this edge. If the edge is a CU-edge with free capacity, then the path again can be continued. If the edge is a CU-edge (a, a') with no free capacity, then we can try to push the flow over a residual constrained edge of some other commodity e' and find an intermediate path (an augmenting cycle) which leads to a', from then on continuing with the original commodity e. This augmenting cycle for the commodity corresponds to finding a reschedule for the communication e' as in Figure 3.3.

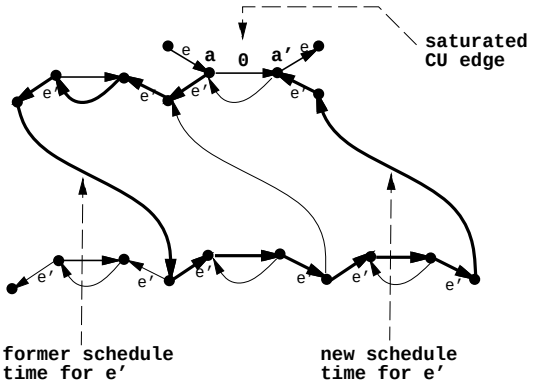

Figure 3.3: A commodity constrained path of label e may also flow along one or several subpaths of some other label e'. This corresponds to scheduling e' some time later, thereby vacating the register file for e.

Theorem 1 *Given a communication graph and constraints on the size of the register files associated with each of the functional units, our algorithm for the equivalent commodity constrained network flow problem obtains a solution if there exists one and gives the schedule for the communications of the data values.*

Theorem 2 *The complexity of the path finding algorithm is $O((m/n)^T)$, where n is the number of functional units, m is the number of edges of the communication graph and T the number of time steps. The pathfinding algorithm has to be invoked exactly m times.*

We do not detail the proof of the correctness and the complexity analysis of the algorithm, as it is rather tedious, but leave the reader with the intuitive ideas outlined above.

This analysis is based on a worst case analysis. In practical examples the number of paths will be much less. This is because the mobility of most communications only stretches over a few time steps and not over the complete range of T. Another practical observation is, that most of the communications can be scheduled at their *lat* times. This is because otherwise one

more register is needed at communication time. This observation can improve the practical performance of the algorithm, in that the augmenting path finding algorithm is tailored to first try and take the last possible path to the target functional unit.

4 Extensions

Above method can be extended to incorporate several other cost measures, such as constraints on the number of buses or the number of ports in each memory.

We sketch the idea for incorporating bus constraints in Figure 4. The idea is simple. Just as the memory edges have capacity constraints, we make sure that all possible communications, which might use a bus at a time step t all flow through a common CC-edge b_t.

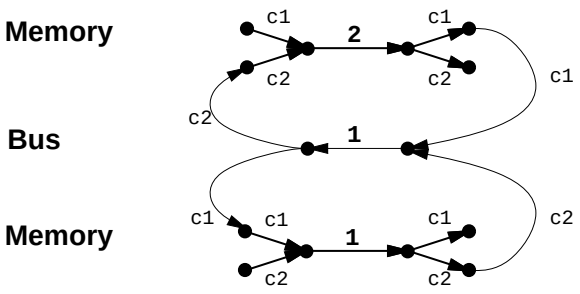

Figure 4: Incorporating bus constraints into the CC-network. All commodity constrained edges flow through one common CU-edge. The capacity is equivalent to the bus constraint, which is 1 here.

Similarly, port constraints for each memory can be incorporated by introducing port edges between the memory edges and the bus edges. Each commodity which can leave the memory at a time t has to flow through the outport edge, the bus edge and through the input port edge of its target memory.

5 Conclusions

We have modeled a general problem in the area of High Level Synthesis, which has been solved by others heuristically or by enumeration techniques, as a special network flow problem. This problem is a Commodity Constrained Network Flow problem. An important graph theoretic formulation is thus presented which could lead to faster algorithms. An algorithm that delivers an exact solution is also shown. All known approaches prior to this work resulted in heuristic algorithms only.

Due to the intelligent enumeration of all possible augmenting paths, we can state an upper complexity bound, which is better than a trivial exhaustive search technique. The model can be extended to incorporate other constraints, e.g. constraints on the number of buses or constraints on the number of input or output ports of the local memory.

6 Acknowledgments

This work has been supported in part by the National Science Foundation under grant MIP 9207267 and by the German Ministry for Education and Research (BMBF) under grant 01-IS-102.

References

[1] H. DeMan, J. Rabaey, P. Six, and L. Claesen. Cathedral-II: A silicon compiler for digital signal processing. *IEEE Design and Test*, 3(6):13–25, 1986.

[2] S. Devadas and R. Newton. Algorithms for hardware allocation in data path synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-8(7):768–781, 1989.

[3] A. A. Duncan and D.C. Hendry. DSP datapath synthesis eliminating global interconnect. *Proceedings of the EURODAC Conference 1993*, pages 46 – 51, 1993.

[4] C. Ewering. Automatic high level synthesis of partitioned busses. *Proceedings of the International Conference on Computer Aided Design*, pages 304–307, 1990.

[5] L. R. Ford Jr. and D. R. Fulkerson. Flows in network. *Princeton University Press*, 1962.

[6] C. T. Hwang, J. H. Lee, and Y. C. Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–475, 1991.

[7] D. Lanneer, M. Cornero, G. Goossens, and H. DeMan. Data routing: A paradigm for efficient data-path synthesis and code generation. *7th High-Level Synthesis Symposium*, pages 17–22, 1994.

[8] M. C. McFarland. Using bottom-up techniques in the synthesis of digital hardware from abstract behavioral descriptions. *23rd Design Automation Conference*, pages 474–480, 1986.

[9] P. G. Paulin and J. P. Knight. Scheduling and binding algorithms for high-level synthesis. *26th Design Automation Conference*, pages 1–6, 1989.

[10] C.-J. Tseng and D.P. Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(3):375–395, 1986.