Power Efficient Module Allocation and Binding

Jui-Ming Chang and Massoud Pedram

CENG Technical Report 95-16

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213)740-4458

July 5, 1995

Power Efficient Module Allocation and Binding

Jui-Ming Chang and Massoud Pedram
Technical Report CENG 95-16
University Of Southern California
Department of Electrical Engineering- Systems
Los Angeles, CA 90089

Abstract

In this report, we investigate the problem of minimizing the total power consumption during the binding of operations to functional units in a scheduled data path with functional pipelining and conditional branching for data intensive applications. We first present a technique to estimate the power consumption in a functionally pipelined data path and then formulate the power optimization problem as a max-cost multi-commodity flow problem and solve it optimally. Our proposed method can augment most high-level synthesis algorithms as a post-processing step for reducing power after the optimizations for area or speed have been completed. An average power savings of 28% has been observed after we apply our method to pipelined designs that have been optimized using conventional techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Low power has become a primary concern for the growing class of portable computer and consumer electronic devices as well as wireless communications and imaging systems. It has thus become necessary to develop estimation and optimization techniques and tools that help achieve low power in these systems. This is a challenging task that requires power modeling, estimation and minimization at all levels of design abstraction from system and behavioral down to logic and layout levels. This paper focuses on the behavioral level.

The behavioral synthesis process consists of three phases: allocation, assignment and scheduling. These processes determine how many instances of each resource are needed (allocation), on what resources a computational operation will be performed (assignment) and when it will be executed (scheduling) [GaDu92] [Stok91] [DeMi94]. Traditionally, behavioral synthesis attempts to minimize the number of resources to perform a task in a given time or tries to minimize the execution time for a given set of resources. With the increasing demand for low power circuits, it has become necessary to modify the three phases of the behavioral synthesis process to minimize the power dissipation.

A number of researches have addressed the problem of minimizing power dissipation during module allocation and binding [RaJh94], scheduling, register allocation and binding [RaJh94] [ChPe95] and by trading off area for power through pipelining or parallelization combined

with voltage scaling [GoOr94] [ChPo92]. The work of [ChPe95] describes a single-commodity network flow solution for the register assignment in a non-pipelined data path. Of particular relevance to the present work is however the work of [RaJh94] where the authors describe a module allocation and binding scheme for low power based on iterative improvement of some initial solution. The authors assume random inputs in a structurally pipelined design (although their approach can also handle non-random input sequences). Their solution technique is heuristic. In contrast, we address the optimization problem in a functionally pipelined data path with conditional branching under arbitrary input statistics and our solution technique is provably optimal without increasing the controller and multiplexor cost or increasing the circuit delay. Our proposed method can also work with most high-level synthesis algorithms (i.e., those which perform scheduling before register or module allocation and binding) as a post-processing stage for reducing power optimization after the optimizations for area or speed have been completed.

The paper is organized as follows. Chapter 2 provides some terminology and gives an overview of the proposed algorithm. In Chapter 3, we describe our switching activity calculation procedure. Chapter 4 describes our power model and the module binding problem. Chapter 5 casts the problem as a max-cost multi-commodity flow problem. Experimental results are reported in Chapter 7 while further discussions, future extensions and concluding remark are given in Chapter 8 and 9, respectively.

Chapter 2

Terminology and Overview

This paper assumes a data flow graph (DFG) with conditional branches and functional pipelining that has already been scheduled with a scheduling algorithm (such as the feasible-scheduling in Sehwa [PaPa88]). The scheduling algorithm takes the data flow graph and the given latency (the number of time steps between two consecutive initiations of the data flow algorithm) and produces a feasible schedule subject to constraints on the total number of available modules (functional units) of each type (cf. Fig. 2.1).

The resulting information can be compactly represented in a *basic allocation table (AT)*. In this table, rows represent the functional units (operators) and columns represent c-steps. A *c-step* refers to a group of concurrent time steps across different pipeline initiations. For example, if a data flow graph is scheduled with latency $L = 3$, then c-step 1 in the associated basic allocation table represents time steps 1 and 4 in the original data flow graph while c-step 2 in that basic AT represents time steps 2 and 5 in the DFG, etc. Furthermore, we annotate each operation in the basic allocation table with its initiation index (that is shown as a superscript on each operation) (cf. Fig. 2.2).

In the previous work [PaPa88], after scheduling and allocation of the functionally pipelined data path, the main optimization tasks on the data path are complete as the circuit speed and the hardware resources have already been determined. Although the binding of different

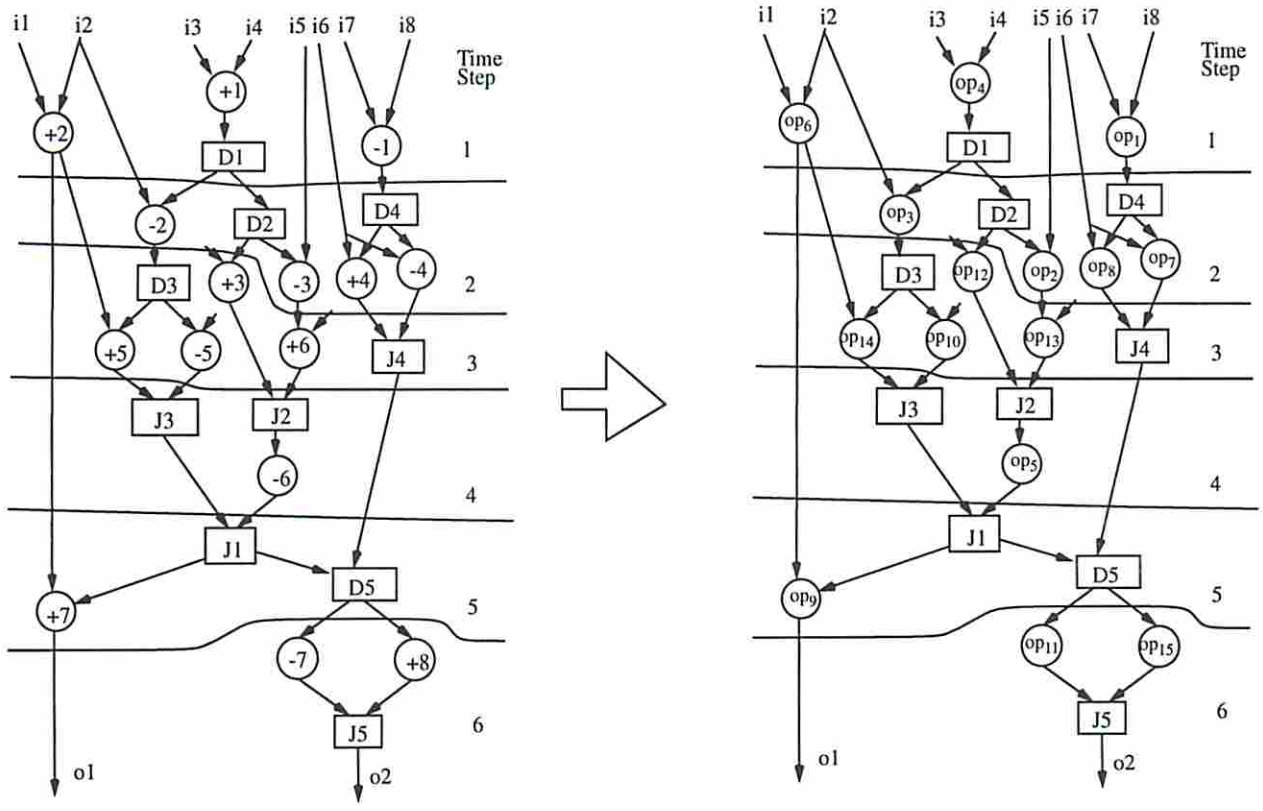


Figure 2.1: An example of an data flow graph with conditional branches and its relabeling

csteps / F.U.	1	2	3
SUB1	-1	(-3,-2)	-5
SUB2	-6	-4	-7
ADD1	+1	+4	(+3,+6,+5)
ADD2	+2	+7	+8

⇨

csteps / F.U.	1	2	3
SUB1	$op_1^{(2)}$	$(op_2,op_3)^{(2)}$	$op_{10}^{(2)}$
SUB2	$op_5^{(1)}$	$op_7^{(2)}$	$op_{11}^{(1)}$
ADD1	$op_4^{(2)}$	$op_8^{(2)}$	$(op_{12},op_{13},op_{14})^{(2)}$
ADD2	$op_6^{(2)}$	$op_9^{(1)}$	$op_{15}^{(1)}$

Figure 2.2: Basic allocation table after scheduling of data flow graph in Fig. 2.1 and its relabeling. Note that the superscript on each operation denotes the pipeline initiation index (or data sample index) of that operation

operations of the compatible type to a set of functional units *(FU's)* is not yet determined, to a first order, this binding does not have much of an impact on the circuit speed or area (see Chapter 8 for a detailed discussion). The binding however has an important effect on the power dissipation as is explained next. Operations that are assigned to a FU in some c-step may be permuted with other compatible operations that are scheduled in the same c-step but are assigned to a different FU. This permutation may have a big effect on the power consumption in the functional units as it changes the sequence of data values going through each functional unit in the data path, thus influencing the switching activity at the inputs of the functional unit.

In a functionally pipelined data path, for a given latency L, there are L c-steps in the allocation table. Operations which share the same FU across consecutive c-steps form a directed cycle of L vertices starting with some vertex in c-step 1 and ending with the same vertex in c-step $L+1$. The total switching activity across all cycles can be determined once we select specific permutations of compatible entries in each c-step (cf. Chapter 3). The problem of minimizing total switching activity is then equivalent to finding the optimal permutation of the entries in each column of the allocation table. This problem can be formulated as a multi-commodity network flow problem and solved optimally (cf. Chapter 5). Since latency of most problems is small even when the data flow graph has a large numbers of levels [1], exhaustive search is also possible, but not desirable. Experimentally, we have observed that the ratio of power consumption in a data path between the power-minimal binding and the area-minimal binding is an average of 0.72.

Our method can be used in conjunction with all other techniques aimed to optimize power consumption at the system or behavioral levels of design. For example techniques of Hyper-LP[ChPo92] that permit a reduction of V_{dd} (such as pipelining or parallelism) may be augmented with the technique developed in this paper to further reduce the power consumption

[1] This is also referred as turn around time or computation time of one input sample for the data flow graph.

without increasing chip area caused by additional multiplexors or increased controller complexity.

Chapter 3

Switching Activity Calculation

Consider two operations that share some FU consecutively (that is there is no other operation that uses this FU in between). Let the two ordered operands for the first operation be x_1 and y_1, the two ordered operands for the second operation be x_2 and y_2, and the outputs of the two operations be z_1 and z_2, respectively. The switching activity at the inputs of the FU for executing these operations is given by:

$$\begin{aligned} switching^{FU}(op_1, op_2) &= \sum_{(x_1,x_2)\in\mathcal{E}} f_{x_1x_2}(x_1, x_2) \times H(x_1, x_2) \\ &+ \sum_{(y_1,y_2)\in\mathcal{F}} f_{y_1y_2}(y_1, y_2) \times H(y_1, y_2) \end{aligned} \tag{3.1}$$

where $f_{x_1x_2}(x_1, x_2)$ is the (word-level) joint probability density function (pdf) [Papo91] of variables x_1 and x_2 and $H(x_1, x_2)$ is the Hamming distance of the binary representations of x_1 and x_2, sets $\mathcal{E}$ and $\mathcal{F}$ are the *legal sets* (domains) of pairs (x_1, x_2) and (y_1, y_2), respectively. More precisely, assume that the n primary input random variables in a single initiation of the pipeline are $a_1, a_2, \ldots, a_n$, and $\mathcal{A} = \{(a_1, a_2, \ldots, a_n)\}$ is the set containing all possible combinations of input tuples, then

$$\mathcal{E} = \{(x_1, x_2) \mid x_1 = x_1(a_1, a_2, \ldots, a_n),\ x_2 = x_2(a_1, a_2, \ldots, a_n),\ \forall(a_1, a_2, \ldots, a_n) \in \mathcal{A}\} \quad \textit{and}$$

$$\mathcal{F} = \{(y_1, y_2) \mid y_1 = y_1(a_1, a_2, \ldots, a_n),\ y_2 = y_2(a_1, a_2, \ldots, a_n),\ \forall(a_1, a_2, \ldots, a_n) \in \mathcal{A}\}.$$

In a functionally pipelined data path, the same algorithm (data flow graph) is initiated every L time steps. We can associate with each different instance of the data flow graph an initiation index. Similarly, the arcs in the data flow graph can also be indexed by an integer tag associated with their data flow instance. Note that the internal arcs of each data flow graph can be converted into only functions of the primary inputs indexed by their initiation index.

Consider a data flow graph with 4 primary inputs a,b,c and d. Suppose that we are interested in calculating the switching activity between two operations $op_1^{(i)}$ and $op_2^{(i+2)}$ that belong to pipeline initiation i and $i+2$, respectively. (We use superscript of an operation to denote its pipeline initiation index or data sample index). Furthermore, assume that the two ordered operands of op_1 are x_1 and y_1 while the two ordered operands of op_2 are x_2 and y_2 and the outputs of the two operations are z_1 and z_2, respectively. If $x_1 = a + b$, $y_1 = c - d$; $x_2 = a \times b$, $y_2 = c/d$, then $x_1^{(i)} = a^{(i)} + b^{(i)}$, $y_1^{(i)} = c^{(i)} - d^{(i)}$; $x_2^{(i+2)} = a^{(i+2)} \times b^{(i+2)}$, $y_2^{(i+2)} = c^{(i+2)}/d^{(i+2)}$.

To utilize equation (3.1), we must have the the joint pdf of the corresponding random variables. Here we create $v_1^{(i)} = x_1^{(i)} = a^{(i)} + b^{(i)}$ and $v_2^{(i)} = x_2^{(i+2)} = a^{(i+2)} \times b^{(i+2)}$; $u_1^{(i)} = y_1^{(i)} = c^{(i)} - d^{(i)}$ and $u_2^{(i)} = y_2^{(i+2)} = c^{(i+2)}/d^{(i+2)}$; $w_1^{(i)} = z_1^{(i)}$ and $w_2^{(i)} = z_2^{(i+2)}$.

If a large number of primary input vectors, say N, is given, then we can calculate the sequence of vector pairs $(v_1^{(i)}, v_2^{(i)})$, $i = 1, 2, 3, \ldots$. If we assume that the sequences of primary inputs $< a^{(1)}, a^{(2)}, \ldots, a^{(N)} >$, $< b^{(1)}, b^{(2)}, \ldots, b^{(N)} >$, etc. are identically distributed, then we can show that each one of the the intermediate sequences $< v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(N)} >$, $< v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(N)} >$, etc. is also identically distributed. We can therefore conclude that the time average can be used to approximate ensemble average, and using the classical frequency interpretation of probability, the joint pdf of v_1 and v_2, $f_{v_1 v_2}(v_1, v_2)$ is approximated by calculating the frequency of occurrence of each (v_1, v_2) pair in the sequence [Papo91]. Similarly, the joint pdf of u_1 and u_2, $f_{u_1 u_2}(u_1, u_2)$ and the joint pdf of w_1 and w_2, $f_{w_1 w_2}(w_1, w_2)$ is calculated.

Consider two sequences of data values $< v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(N)} >$ and $< v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(N)} >$

going to the inputs of a Mux. Switching activity at the inputs of the Mux is given by

$$\begin{aligned} switching^{Mux}(v_1, v_2) &= switching(v_1) + switching(v_2) \\ switching(v_1) &= \frac{1}{N-1} \sum_{t=1}^{N-1} H(v_1(t), v_1(t+1)) \end{aligned} \tag{3.2}$$

The calculation of the switching activity on the inputs of the Mux can be done at the same time as the calculation of the joint pdf $f_{u_1 u_2}(u_1, u_2)$ in the previous paragraph.

Using above procedures, we only need a single scan through the input vectors to obtain all of the switching activities and joint *pdf's*. The run time is thus proportional to the number of the primary input vectors applied to the system.

We need the sequence of input vectors for the above switching activity calculation procedure to work. This sequence may be a "short" (in hundreds of vectors) sequence of typical data stream obtained by statistical sampling [BuNa93] or may be a "long" (in tens of thousands of vectors) obtained from a dynamic execution trace for a program/application data runs on the target chip. We make no assumption about the data statistics, hence, our technique is applicable to both DSP chips where the data tends to exhibit a pseudo-random white noise behavior and ASICs or general-purpose processor chips where the data may exhibit any probability distribution. Furthermore, we only assume that the sequence of input vectors is identically distributed. A similar assumption is commonly used in other fields like digital modulation, communication system performance evaluation and spectral analysis. This assumption is empirically justified and allows one to use formal methods to analyze the input streams [Gall68] [ViOm79].

In most designs (and this is not limited to DSPs), designers can provide either a "short" or a "long" input sequence which represents their application data. We exploit this data sequence to our advantage.

Chapter 4

Module Binding with Minimum Power Dissipation

4.1 Power dissipation of a functional unit

We assume that the dynamic power dissipation in a functional unit when it executes op_2 after op_1 is given by a simple equation as follows:

$$P_{FU} = \frac{1}{2}\alpha \cdot V^2 \cdot f \cdot switching^{FU}(op_1, op_2) \tag{4.1}$$

where V is the supply voltage, f is the clock frequency, and the proportionality constant α (which represents the physical capacitance of the functional unit) is calculated for each functional unit using circuit or gate level simulation [Deng94][BuNa93] and curve fitting. Obviously this proportionality constant depends on the module type, input data width, technology and logic style used and internal module structure. Equation (4.1) is the basis of all macro-modeling techniques for power estimation and has been used in the work of [PoCh91][LaRa94][SvLi94] [MeRa94]. Power estimation accuracies of 10-15% have been reported for this model in [LaRa94].

To be more specific, we present some results for a set of datapath modules implemented in a 1 μ technology in Table 4.1. Here the first column gives the functional unit name, the second column gives the actual power dissipation obtained by gate-level simulation (using 10,000 input

Circuit	$Power_{actual}$ (μWatts)	α	switchings	$Power_{est}$ (μWatts)	error %
add8	1019.39	9.46	0.4199	993.06	2.58
add16	2082.70	18.91	0.4282	2024.31	2.80
add32	3980.91	37.82	0.4073	3851.02	3.26
mult8	10949.51	100.16	0.4081	10218.82	6.67
mult16	44076.52	400.64	0.4094	41005.50	6.96
mult32	166235.89	1602.56	0.3904	156409.86	5.91
Mux8: 2 to 1	240.50	1.98	0.5	247.5	3.11
Mux16: 2 to 1	481	3.96	0.5	495	7.6
Mux32: 2 to 1	962	7.92	0.5	990	2.9
Mux8: 4 to 1	680.32	5.58	0.5	697.5	2.5
Mux16: 4 to 1	1360.64	11.16	0.5	1395	2.52
Mux32: 4 to 1	2721.28	22.32	0.5	2790	2.52
Average	-	-	-	-	4.11

Table 4.1: Data-Path Circuits and their simulation results

vectors), the third column gives the α parameter obtained by curve-fitting, the fourth column gives the average switching activity per input line of the module and finally the last column gives the estimated power dissipation obtained from equation (4.1). V= 5 volts, f=20 Mhz and α is in units of pF. The table shows that equation (4.1) provides power estimation which are an average only 4.11% in error (compared to $Power_{actual}$).

4.2 A functionally pipelined data path

In this work, the data path is assumed to be functionally pipelined. If the operations that share the same functional unit belong to different pipeline initiations, then the joint pdf's of any two random variables belonging to different pipeline initiations will account for both spatial and temporal dependencies [1].

[1]Spatial correlation between x & y refers to the dependence of $x(t)$ on $y(t)$ or vise versa while the temporal correlation on the line x refers to the dependence of $x(t+i)$ on x(t).

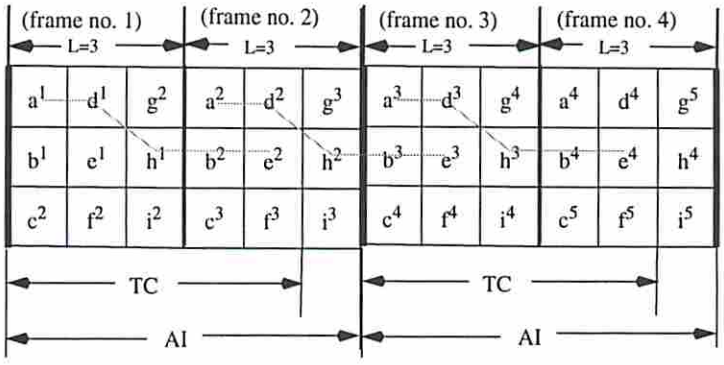


Figure 4.1: The definition of AI and TC. Note that the superscript on each operation denotes the pipeline initiation index (or data sample index)

Definition 4.1 *In Fig. 4.1, suppose we have a sequence of operations* $[a^{2i-1} \rightarrow d^{2i-1} \rightarrow h^{2i-1} \rightarrow b^{2i} \rightarrow e^{2i}]$ *(for* $i = 1, 2, \ldots$*) that share the same functional unit. The time span of this sequence is defined as TC, which is equal to 5 time steps. The alignment interval of this sequence, AI is defined as the number of time steps required to allow the execution of this sequence periodically. In this example, AI is 6, which is equal to two (time) frames. We use time frame to refer to a sequence of c-steps of length L where L is the latency of the pipeline.*

Lemma 4.1 *Consider a data flow graph which executes with latency L for the functional pipeline. Suppose there are k operations,* op_1, op_2,..., op_k *that share some functional unit consecutively, and that the time span (in terms of the number of time steps) of the sequence of operations is TC. To sustain this sequence of operations on the FU in question periodically across many frames in the pipeline, the alignment interval AI for this set of k operations must be an integer multiple of L which is larger than or equal to TC. Proof follows from definition of AI.*

In the past, most of the work on functionally pipelined data path has focused on the treatment of only a frame of length L. For the problem of sharing a FU among many operations which are scheduled in different c-steps, one should however consider a sharing chain with length larger than L since two operations which share a FU may belong to different c-steps

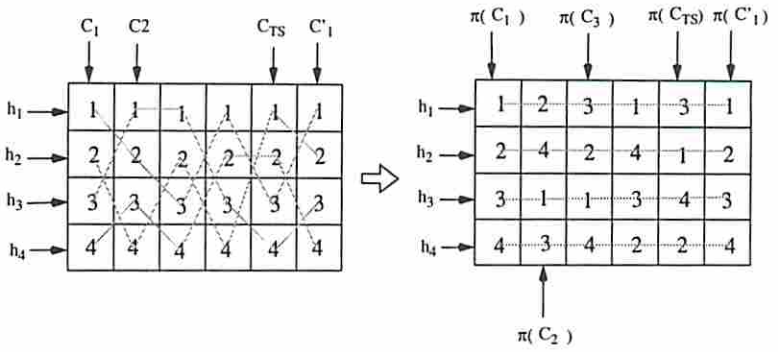


Figure 4.2: Valid binding resulting from permutation of the column entries in the *AT*

in *different time frames* (cf. Fig. 4.3). This results in little reduction in power consumption, but creates large sharing chains and thus leads to large controller complexity and multiplexor cost which tend to offset power reduction due to sharing the FU across multiple frames. For this reason, we have decided to consider FU sharing only for operations in the same time frame. If we ignore the switching activity resulting from the last operation of previous pipeline initiation to the first operation of the current pipeline initiation (we call this the *boundary switching*), then the optimal FU assignment in the last pipeline initiation will also serve as the optimal FU assignment for the current pipeline initiation. The reason is that the optimization process in the two initiations uses the same set of parameters (that is, the average switching activities between consecutive operations are identical in both pipeline initiations). We make this statement more precise in Appendix A.

4.3 The optimization problem

The problem is to find the power optimal way of binding operations to a set of compatible modules. Each binding solution corresponds to a permutation of the column entries of the *AT*. The solution also specifies multiple chains of operations where each chain denotes all operations that are consecutively executed on a functional unit. The total switching activity

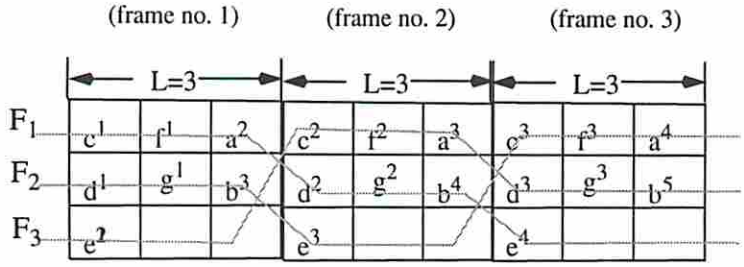


Figure 4.3: Three whole chains of 9 time steps composed of 3 optimal sub-chains. When we consider FU sharing within one time frame only, then the sharing solution within one frame (say frame # 1) will be replicated across all frames.

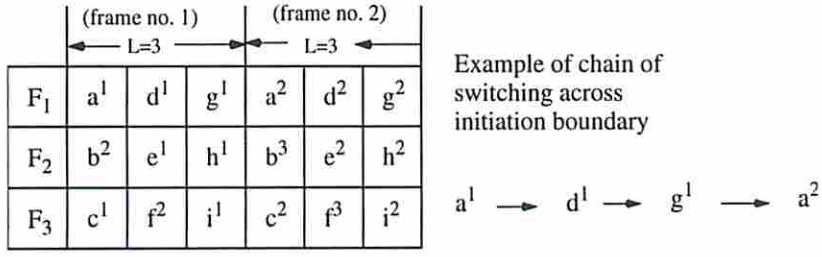


Figure 4.4: Boundary switching between two frames

of the chain consists of the sum of switching activities between two consecutive entries, plus the switching from the last entry of the row in the current frame to the first entry of the same row in the next frame of length L (see Fig. 4.4). The total binding cost is the sum of the costs of all chains. Because each operation is executed on exactly one module, these chains must be node disjoint.

Definition 4.2 *A basic allocation table consists of m rows corresponding to m compatible modules and L columns corresponding to L c-steps. An extended allocation table (EAT) is obtained by concatenating the first column of the basic allocation table after its rightmost column thus obtaining a new table with $L+1$ columns where the first and the last columns are identical (cf. Fig. 4.5).*

□

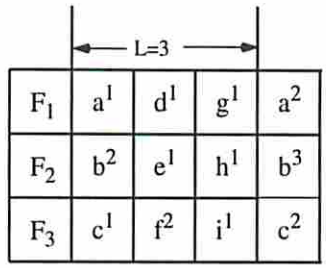


Figure 4.5: Extended allocation table (EAT)

The optimization problem is then equivalent to finding the optimal way to permute the elements in each column (except the first and the last column in the EAT) for the rows corresponding to the set of compatible modules and minimizing the switching cost in all rows.

Definition 4.3 *The requirement that the* $(i, 1)th$ *entry of the* EAT *be equal to the* $(i, L + 1)th$ *entry of the table for* $i = 1, \ldots, m$ *will be referred as the* inter-frame binding constraint. *This condition is imposed to gauranttee the cyclic nature of the execution on the functionally pipelined datapath without having to incur a large cost in terms of controller complexity and size of the MUX's.*

Chapter 5

Multi-Commodity Flow Formulation

At the end of the Chapter 4.2, we described the optimization of the total switching activity using the EAT. In the following, we cast the optimization problem as a *Max-Cost Multi-Commodity Flow* problem. The condition that makes the original problem hard is that we must meet the inter-frame binding constraint. If we did not have to meet the inter-frame binding constraint, a simple *Max-Cost Flow* would give the optimal solution [ChPe95].

Suppose there are m rows and n columns ($n = L + 1$) in the EAT. In the simplest case, no entries in the EAT are empty. In general, there may however exist some empty columns. During scheduling and allocation of a functionally pipelined data path, we use the minimum possible number of modules and hence the feasible scheduling results in an allocation table which contains at least one full column. [1] Suppose this column is at c-step i in the basic AT, we can rotate columns of this basic allocation table until that full column occupies the first position in the table. Then we construct the EAT from the new AT by augmenting it with a rightmost column which contains the entries of the first column in the new AT in the next frame of length L. Fig. 5.1 shows an example of rotating a basic AT and obtaining the new EAT where first and last columns are full. This basic AT is obtained by applying feasible scheduling [PaPa88] on DFG shown in Fig. 6.1 with latency 3. If the first and the last columns

[1] The case where we have more compatible modules of the same type than we actually need, can be handled without much difficulty, but is skipped here for the sake of brevity.

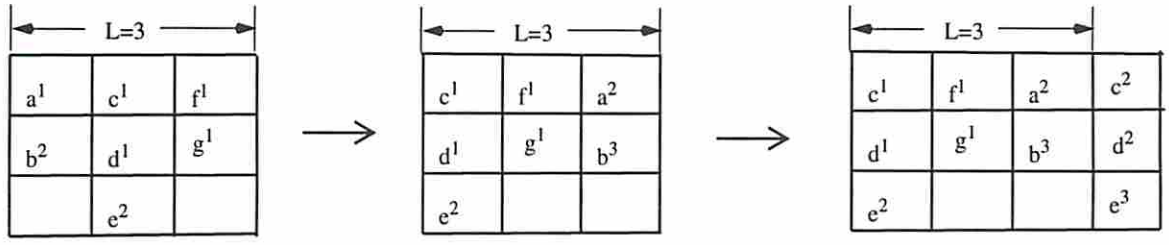


Figure 5.1: Rotating the basic AT and obtaining the new EAT

in the EAT have empty entries, then the flow on the network which will be constructed from this EAT will no longer represent all of the feasible permutations. This is the reason for the EAT construction given above.

Let N_H=$(V, E, s_1, s_2, \ldots, s_m, t_1, t_2, \ldots, t_m, C, K)$. There is a vertex $v \in V$ for each nonempty entry in the EAT (with m rows and n columns). We will refer to V as internal node set of N_H. We add a column of vertices s_i, $i = 1, \ldots, m$ in front of the vertices corresponding to the operations in the first column of the EAT. We also add a column of vertices t_i, $i = 1, \ldots, m$ after the vertices corresponding to the operations in the last column of the EAT. There is an arc connecting s_i to the vertex corresponding to the $(i, 1)th$ entry of the EAT and an arc connecting vertex corresponding to $(i, n)th$ entry of the EAT to vertex t_i, $\forall\ i = 1 \ldots m$. The vertices are then levelized (with s_i vertices at layer 0 and t_i vertices at layer $L + 2$). There are arcs from all of the vertices in layer i to all of the vertices in layer $i + 1$, $\forall\ i = 1 \ldots n - 1$. Besides, there will be arcs from all of the vertices at layer $i - 1$ to all of the vertices at layer $i + 1$ if there exists some empty entry in column i of the EAT, $\forall\ \ i = 1 \ldots n$. If there are empty entries in both columns $i - 1$ and i in the EAT, we add arcs from all vertices at layer $i - 2$ to vertices at layer $i + 1$ in N_H, and so on.

The capacity function K is 1 for every arc $e \in E$. The cost of all arcs incident on s_i or t_i is 0. All other internal arcs have cost $C(u, v) \geq 0$ which is equal to $H - \lfloor M \times switching^{FU}(op_u, op_v) \rfloor$, where M is a large number to magnify the switching activity into an integer and H is a sufficiently large integer that makes the resulting costs for

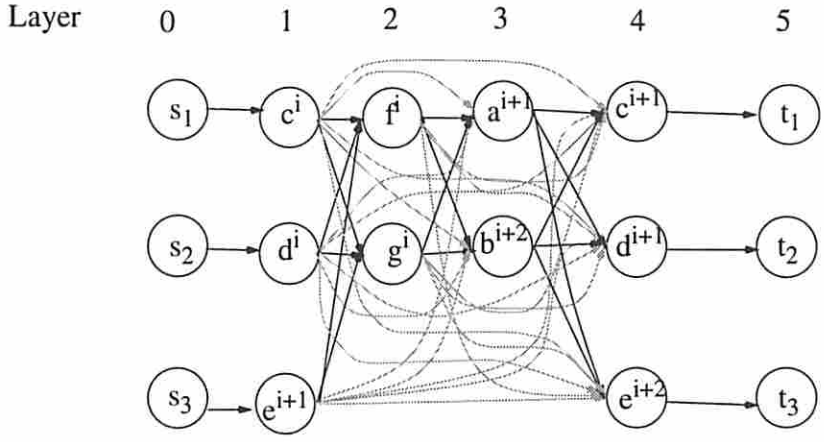


Figure 5.2: Network N_H construct from the new EAT shown in Fig. 5.1. Dark edges represent edges from level i to $i+1$ while light edges represent edges from level i to $j > i+1$

all of internal arcs satisfy the *triangular inequality.* [2] Let $\beta(\gamma)$ be the maximum (minimum) of $\lfloor M \times \ switching^{FU}(op_u, op_v) \rfloor$ over all u, v. H is any integer that satisfies $H \geq 2\beta - \gamma$. This is needed to ensure that the network flow algorithm covers all of the vertices in N_H. Fig. 5.2 shows the network constructed from the new EAT shown in Fig. 5.1. The demand function D is defined as follows: $D_i(v) = 0$, $\forall v \in V$ & $\forall i = 1, \ldots, m.$; $D_i(s_j) = \begin{cases} -1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$; $D_i(t_j) = \begin{cases} +1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$. This function captures the inter-frame binding constraint of Chapter 4.3.

Definition 5.1 *[Leng90]*

Min-Cost Multi-Commodity Flow:

- *Instances: A directed graph G=(V,E), edge capacity K(e) $\in R_+$ and edge costs C(e) $\in R$ for e $\in E$, demands $D_i(v) \in R$ for all vertices v $\in$ V and for m commodities i = 1, ..., m.*

- *Configurations: All sequences of edge labeling f_i: $E \rightarrow R_+$, i=1, ..., m*

[2] That is, $C(i,j) \geq 0$, $\forall$ (i,j) and $C(i,j)+C(j,k) \geq C(i,k)$, $\forall$ internal arcs (i,j),(j,k) and (i,k) (if the arcs exist).

- *Solutions: All sequences of edge labeling that satisfy the following constraints:*

  1. *Capacity Constraints: For all* $e \in E$, $\sum_{i=1}^{m} f_i(e) \leq K(e)$

  2. *Flow-Conservation Constraints: Define the net flow of commodity* i *into vertex* v *to be*

$$f_i(v) = \sum_{e:\, u \to v} f_i(e) - \sum_{e:\, v \to w} f_i(e)$$

  Then, for all $i = 1, \ldots, m$ *and* $v \in V$, *we have* $f_i(v) = D_i(v)$

- *Minimize:* $C(f) = \sum_{e \in E}(C(e) \times \sum_{i=1}^{m} f_i(e))$

In [Stok91], a multi-commodity flow formulation of the register allocation problem in a cyclic data flow graph is proposed. In this paper, we propose a multi-commodity flow formulation of the module allocation and binding in a functionally pipelined data flow graph. In addition to this, our network structure is quite different from that of [Stok91] due to empty entries that we have in the EAT which leads to cross-over edges (arcs that connect non-adjacent layers in the network). In [Stok91], the network for solving the register allocation problem in a cyclic DFG has a node for each color (or register) at each time step. The resulting network thus uses a dummy node for every time step boundary even when no variable has to be stored there. Also the purpose is to only use this network to consider all feasible permutations and therefore the cost of each arc is one (including the arcs to and from dummy nodes). We could not minimize the total switching activity by simply replacing the cost on each arc in their network by the actual switching activity. The reason is that the switching activity to and from the dummy nodes could not be defined and there is no way to make the resulting network flow represent the total switching activity. In our network we do not have any dummy nodes and each arc has its own (possibly) distinct cost. Eliminating the dummy nodes however creates the situation that some internal node may not be covered in the max-cost multi-commodity flow. We therefore enforce the triangular inequality on the costs of all arcs (and this is always

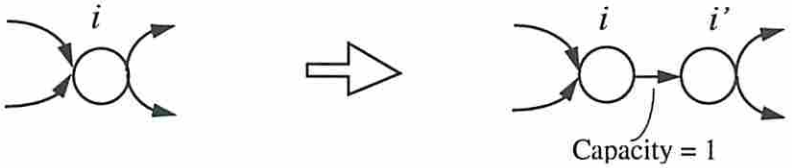


Figure 5.3: The node splitting process on node i

doable) in order to ensure that the max-cost flow covers all of the internal vertices. This enforcement has no impact on the optimality of our solution as will be proved in Theorem 5.1.

Definition 5.2 *In the max-cost multi-commodity-flow problem, we maximize the total cost of the flow $C(f)$ in the network while satisfying constraints 1 and 2 given in definition 5.1.*

The extra vertices s_i and t_i, $i = 1..m$, will serve as the sources and sinks of commodity i, respectively. We ship from s_i one unit of i and sink one unit of i at sink t_i. To ensure that the flow paths are node-disjoint, we apply a node splitting technique on N_H as illistrated in Fig. 5.3. After applying the node splitting on the internal node set of N_H, we obtain N'_H=(V', E', $s_1, s_2, \ldots, s_m$, $t_1, t_2, \ldots, t_m, C, K$). We will refer to V' as the internal node set of N'_H.

The flow with value m on the new network N'_H gives m node disjoint paths; each path starting from source s_i and ending at sink t_i, for all i. We conduct the *max-cost* m-commodity flow on N'_H, which minimizes the total switching activity while satisfying the inter-frame binding constraint. The network formulation provides the exact solution to the original problem as shown by the following theorem.

Theorem 5.1 *A max-cost multi-commodity flow of value m on the network N'_H gives the minimum total power consumption for the m compatible modules in the circuit while satisfying the inter-frame binding constraint as defined in Definition 4.3.*

Proof: For an EAT with m rows, the total cost on N'_H is $\sum_{e \in E'}((\sum_{i=1}^{m} f_i(e)) \times C(e))$, which is a linear function of the *"Total Saved Power"*. The reason is that

$$\sum_{e \in E'}((\sum_{i=1}^{m} f_i(e)) \times C(e)) = \sum_{e \in E'}((\sum_{i=1}^{m} f_i(e)) \times [H - M \times switching^{FU}(e)]) =$$

$$H \times \sum_{e \in E'} \sum_{i=1}^{m} f_i(e) - M \times \sum_{e \in E'} \sum_{i=1}^{m} f_i(e) \times switching^{FU}(e).$$

In our specially constructed network, $\sum_{i=1}^{m} f_i(e)$ in every arc e has value either zero or one. The $\sum_{e \in E'} \sum_{i=1}^{m} f_i(e)$ term is a constant equal to $|V'| + m$ for all possible valid network flows that cover the internal node set of N'_H. Therefore when we maximize the total cost for a given flow value of m in N'_H (flow of one unit on each commodity supplied by s_i and demanded by t_i, $\forall\ i = 1, \ldots, m$), we are minimizing the total switching activity (and hence power dissipations according to equation (4.1)) for m compatible modules. The inter-frame binding constraint is satisfied in all feasible flows because of the demand function and the flow conservation constraint.

□

Although our network is constructed differently from [Stok91], a similar method for solving the remaining step can be used after the multi-commodity network flow problem is translated into a LP.

Since we are considering the functionally pipelined data path where the latencies of most pipeline design are quite small, exhaustive search on the EAT can also give the optimal solution while meeting the inter-frame binding constraint in a very short time.

Chapter 6

A Detailed Example

In this Chapter, we present a detailed example which is synthesized from the DFG shown in Fig. 6.1 with a latency of 3. The resulting EAT (obtained after rotating the basic AT) was shown in Fig. 5.1 while network N_H was depicted in Fig. 5.2.

The power cost for pairs of operations (that is, $H - M \times switching^{FU}$ where H and M were set to 1000) is given in Table 6.2.

Using the max-cost multi-commodity flow algorithms, the result of min-power module binding are shown in Fig. 6.2(a). The total power dissipation for this binding solution is calculated as follows. There are three chains in the EAT; The power consumption of the $FU's$

Module	α
add16	18.91
mult16	400.64
Mux16: 2 to 1	3.96
Mux16: 4 to 1	11.16

Table 6.1: different α's obtained from Simulation

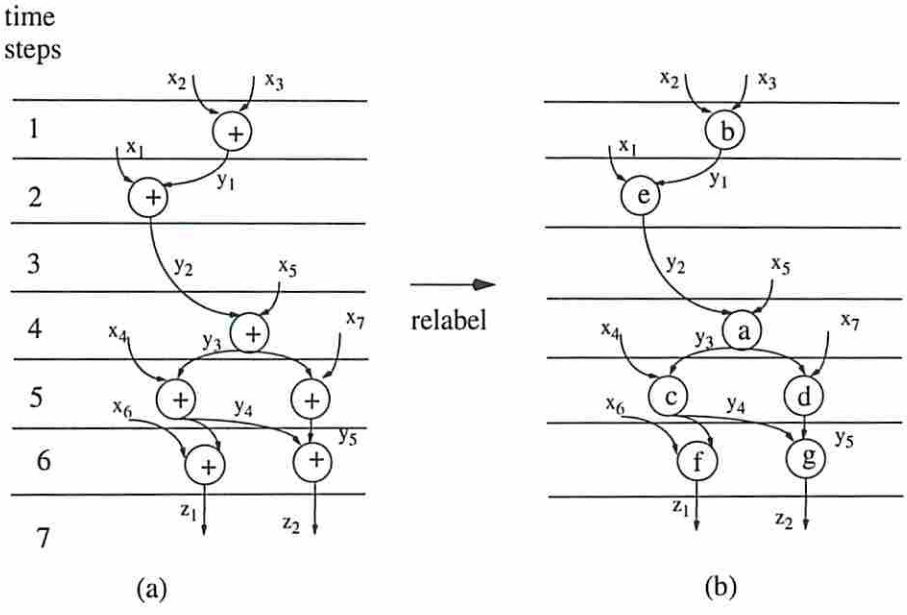


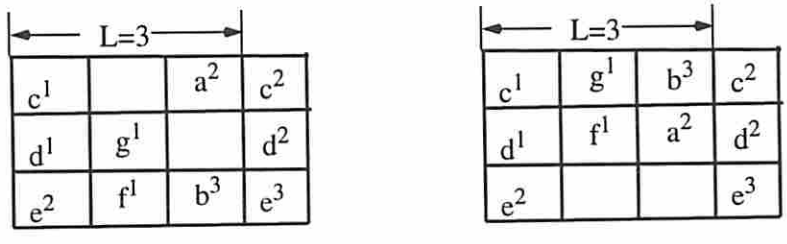
Figure 6.1: A very simple DFG whose basic AT is shown in Fig. 5.1

	f^1	g^1	a^2	b^3	c^2	d^2	e^3
c^1	515	445	537	539	467	588	754
d^1	403	743	511	700	458	566	628
e^2	551	750	543	423	527	476	401

	a^2	b^3	c^2	d^2	e^3
f^1	413	574	416	696	482
g^1	652	403	718	600	572

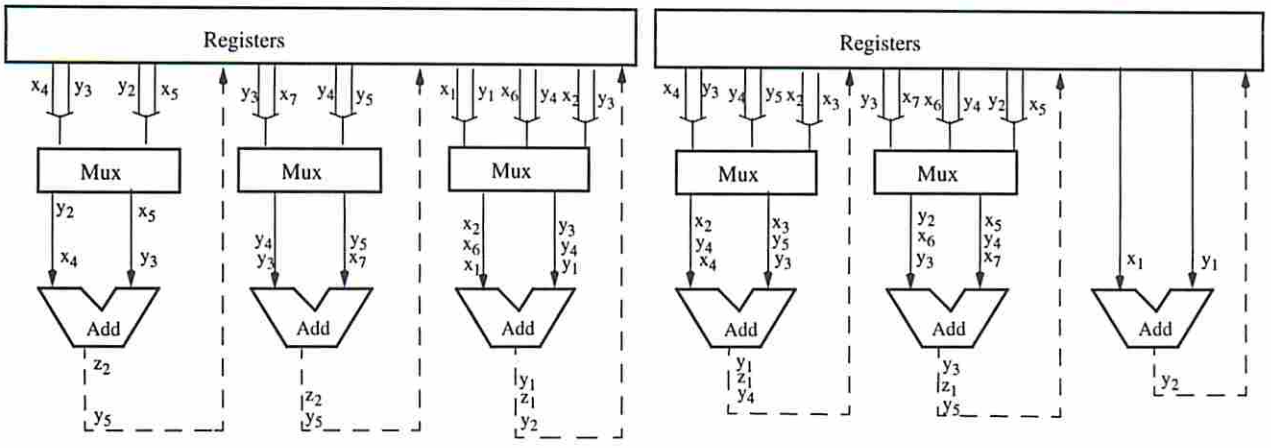
	c^2	d^2	e^3
a^2	749	701	765
b^3	490	404	800

Table 6.2: Cost Matrix for arcs in the network of the second example



(a) Min-Power Binding (b) Max-Power Binding

Figure 6.2: (a)Min and (b)Max power binding for the allocation table of Fig. 5.1.



(a) Min-Power Implementation (b) Max-Power Implementation

Figure 6.3: RTL structure for the above example

shared by each chain of operations is given by:

$$
\begin{array}{lll}
\textit{chain } c^1 \rightarrow a^2 \rightarrow c^2 & 18.91 \times 250 \times [(1-0.537)+(1-0.749)] & = 3375.43(\mu W) \\
\textit{chain } d^1 \rightarrow g^1 \rightarrow d^2 & 18.91 \times 250 \times [(1-0.743)+(1-0.600)] & = 3105.96(\mu W) \\
\textit{chain } e^2 \rightarrow f^1 \rightarrow b^3 \rightarrow e^3 & 18.91 \times 250 \times [(1-0.551)+(1-0.574)+(1-0.800)] & = 5082.06(\mu W) \\
P_{total} & & = 11563.39(\mu W)
\end{array}
$$

where the factor 250 comes from $\frac{1}{2}V^2 \cdot f$ where $V = 5$volts and $f = 20\ Mhz$. This result is identical to the one obtained by exhaustive search. The exhaustive search also finds the max-power module binding which is shown in Fig. 6.2(b) has a total power dissipation of 17699 μW. The ratio of power consumption of the best binding to that of the worst binding is 0.65. The resulting RTL structures after the scheduling, allocation and binding are shown in Fig. 6.3(a) and (b), respectively.

Chapter 7

Experimental Results

In Table 6.1, we are showing the values of α parameter for 16-bit adder, multiplier, 2 to 1 Mux and 4 to 1 Mux, this is all the data we need since our benchmark circuits have a datapath width of 16-bits and a latency of less than 4. Table 7.1 gives P_{total}, the total power dissipation in each circuit after scheduling, allocation and binding, that is ,

$$P_{total} = \sum_{FU \in DFG} P_{FU}$$

where P_{FU} was given in equation (4.1). Values of α in equation (4.1) are read from Table 6.1, while values of $switching^{FU}(op_1, op_2)$ are obtained from the binding solution as detailed in Chapter 4. Table 7.2 gives P'_{total} which is $P_{total} + P_{Mux's}$; Again α for the Mux is read from Table 6.1 while switching activity for Mux is calculated from the binding solution.

We performed feasible scheduling and our new method on various other benchmarks including a second example shown in Fig. 2.1 taken from [PaPa88], AR Filter, Elliptical Wave Filter[GeEl92], 2nd order Adaptive Transversal Filter [Hayk91], Robotic Arm Controller, Differential Equation Solver [CaWo91], and Discrete Cosine Transform. The power dissipation results are given in Table 7.1 and 7.2. The latency used in each benchmark is denoted in the 2nd column of the tables. In our experiment, we also generate all possible bindings (all are area-minimal bindings) from the same basic AT for each DFG using feasible scheduling algorithm [PaPa88]. From these tables, we can see that the ratios of P_{total} for minimum-power

Benchmark	Lat.	max. pwr. μW	avg. pwr. μW	min. pwr. μW	min/max %	min/avg %
Example # 1	3	17699	14050	11563	65.33	82.29
Example # 2	3	20947	17331	12404	59.21	71.57
AR Filter	3	1060631	776556	504876	47.60	65.01
EW Filter	3	532885	400576	303027	56.86	75.64
2nd AT Filter	2	310544	261601	175637	56.55	67.13
Robotc	2	1216329	1076411	553346	45.49	51.40
Diff Equation	2	316896	268592	204980	64.68	76.31
FDCT	3	1019158	822642	623237	61.15	75.77
Average	-	-	-	-	56.88	70.64

Table 7.1: P_{total} (power dissipations in $FU's$)

binding to maximum-power and average-power bindings are 56.88% and 70.64%, respectively. Even after including the power dissipation due to $Mux's$ (which our algorithm does not directly attempt to minimize), we obtained power saving ratios of 58.97% and 72.07%, respectively.

Benchmark	Lat.	max. pwr. μW	avg. pwr. μW	min. pwr. μW	min/max %	min/avg %
Example # 1	3	20489	16637	13948	68.08	83.83
Example # 2	3	26527	22911	17984	67.79	78.49
AR Filter	3	1071656	787401	515721	48.12	65.49
EW Filter	3	547825	415516	317967	58.04	76.52
2nd AT Filter	2	314324	265381	179417	57.08	67.61
Robotc	2	1228974	1089086	565991	46.05	51,97
Diff Equation	2	319371	271068	207455	64.96	76.53
FDCT	3	1031488	834792	635567	61.61	76.13
Average	-	-	-	-	58.97	72.07

Table 7.2: P'_{total} (power dissipations in $FU's + Mux's$)

Chapter 8

Discussions and Extensions

The main idea presented here is to permute the compatible operations during the module binding step so as to minimize the switching activity at the inputs of the functional units in a functionally pipelined design. This permutation takes place over a time frame of L c-steps, and therefore, the number of multiplexors doesn't vary by much; and in any case, the power dissipation in Mux's is significantly lower than that in functional units as seen by the relative magnitude of α parameters in Table 4.1. Hence, we can safely assume that P_{Mux} will vary minimally as a function of the binding solutions and when it does, it causes little change in P_{total}. This is also seen in Tables 7.1 and 7.2, where $\frac{P'_{total}}{P_{total}}$ is only 1.072 on average and in Fig. 6.3 where the two binding solutions use different number of $Mux's$, but P_{Mux} is roughly the same for both solutions.

The number of registers (obtained by performing register allocation on a functionally pipelined data path by a program such as REAL [KuAl87]) will also not change as the life time of data values in the data flow graph will not change after the permutation (see Chapter 4.3). The circuit speed will not change as we only permute compatible operations in the same c-step (same column in the EAT) and the number of c-steps is not altered. The only impact of this permutation is to vary the interconnect structure of the design in some undetermined fashion (for worse or for better). Most high level synthesis programs [KuAl87] however ig-

nore this factor (e.g. wiring topology and length) during module allocation and binding as it is very difficult to accurately estimate it without having some information about the circuit floorplan. If high level synthesis and floorplanning are performed concurrently and interactively, then some information about the impact of permuting operations on the wiring length may be known. This additional information however complicates the problem since the cost of each operation binding becomes highly dependent on other operation bindings and thus an incremental, greedy heuristic must be used to find a power-efficient module binding solution accounting for both power and interconnect costs.

Suppose there is a *multi-cycle operation* x in the EAT which spans from c-step i to c-step j. The only possible arcs to and from this operation are from operations scheduled before c-step i and those after c-step j. All of the other steps in constructing the network and solving the multi-commodity flow problem remain unchanged.

One way to handle *conditional branches* is first to apply the node coloring technique of [PaPa88] to find groups of mutually exclusive operations and then create an equivalent operation to represent each such group which are assigned to the same FU in the same c-step. The switching activity calculation is then performed on the resulting EAT by taking this exclusiveness into account. The max-cost multi-commodity flow step remains as before (See Appendix B for details).

For *register binding*, if we consider only the sharing of registers among variables within one frame, we can first use the left edge algorithm to obtain an initial solution , then use our proposed technique for doing module allocation on DFG with multi-cycle operations to solve the minimum power register allocation problem. [1] If a variable's life time is longer then the pipeline initiation latency L, then multiple instances of the same variable will appear in frame $[1, L]$. In fact, for a variable with a life time interval $[a, b]$, there will be $\lceil (b-a)/L \rceil$ instances in the frame. Also, if a variable's life time interval crosses the frame boundary, we must meet

[1] Since a variable's life time possibly spans multiple c-steps, it has to be modeled as a multi-cycle operation.

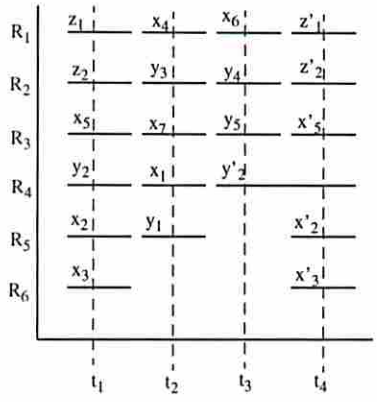


Figure 8.1: Minimum power register allocation obtained by our method

the inter-frame binding constraint by using the same register to store that variable across the frame boundary. If we simply wanted to find a register allocation that meets the inter-frame binding constraint without regard to the power dissipation then the method presented in [Stok91] suffices. However, a network construction similar to the one that we proposed in Chapter 5 for the module allocation problem should be used to minimize the power while meeting the inter-frame binding constraint.

Chapter 9

Conclusion

This paper presented a new method to calculate the switching activity of modules in a functionally pipelined data path with conditional branches based on the assumption that a large number of primary input data vectors are given. The power consumption model for a functionally pipelined data path was presented and its properties were explored. The original power optimization problem for the functionally pipelined data path was then formulated as a multi-commodity flow problem and solved optimally without increasing the area or delay of the data path or the controller complexity compared to the pipelined data path design before using our new method. Both techniques cover very general class of applications and are also very practical for larger problem sizes with complicated control flow. Experimental results demonstrated that the above methods can reduce the power consumption substantially.

Appendix A

Why consider sharing only within a frame

A partial chain (or sub-chain) is a subset of a (complete) chain if it starts at column 1 and ends at column L. We argue that in order to find the power optimal FU binding solution, one need not consider complete chains (sequence of operations sharing the same FU across multiple initiations of the pipeline), but only partial chains (sequence of operations sharing the same FU within exactly one pipeline initiation). Intuitively, the reason is that optimal complete chains of length $m \times L$ consist of the same set of optimal partial chains (sub-chains) of length L. That is, if C denotes the set of optimal sub-chains in $[1, L]$, then every optimal complete chain will only consist of elements of C. Fig. A.1 shows an example where the set of optimal FU assignments (set of optimal partial chains) in each pipeline initiation consists of three such partial chains as follows: $[c \rightarrow f \rightarrow a]$, $[d \rightarrow g \rightarrow b]$ and $[e]$, and the dotted lines show 3 complete (whole) chains with period of $3 \times L$ (9 time steps), which consist of the three partial chains. For example, a complete chain in Fig. A.1 is $[c^1 \rightarrow f^1 \rightarrow a^2 \rightarrow d^2 \rightarrow g^2 \rightarrow b^4 \rightarrow e^4]$. Here, each of the three complete chains contains 7 operations across 9 time steps, this implies that a physical module must be shared by 7 operations. If we use multiplexors to implement this, the controller complexity and the power consumed in the Mux's will be large. In practice, the power savings obtained by optimizing the boundary switching will be more than offset by

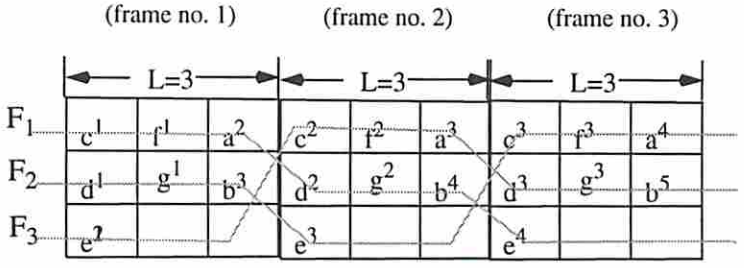


Figure A.1: Three whole chains of 9 time steps composed of 3 optimal sub-chains. When we consider FU sharing within one time frame only, then the sharing solution within one frame (say frame # 1) will be replicated across all frames.

the increase in controller complexity and the power dissipation in the Mux's.

If we consider the case that AI is exactly equal to L, the set of the optimal partial chains (sub-chains) is the same as that in the previous case with $AI = 3 \times L$. This implies the we reach the same optimal solution as that of using the $AI = 3 \times L$ except that the boundary switching is not globally optimized. If we unroll the whole chain of $AI = L$ three times, we obtain one equivalent longer chain of $AI = 3 \times L$: $[c^1 \rightarrow f^1 \rightarrow a^2 \rightarrow c^2 \rightarrow f^2 \rightarrow a^3 \rightarrow c^3 \rightarrow f^3 \rightarrow a^4]$. The new set of equivalent complete chains of $AI = 3 \times L$ are composed of exactly the same set of optimal sub-chains, except that the boundary switching between adjacent initiations is different. But in this case, at most 3 operations will be shared by any module. We don't have to use a bigger Mux than that used by Sehwa [PaPa88], neither do we increase the controller complexity. For this reason, from now on, we only consider the case that AI is exactly equal to L.

Appendix B

Handling conditional branches

Two operations can share the same module only if they are not scheduled at the same c-step, However, two or more operations that are mutually exclusive, even when they are scheduled in the same c-step, they may still share the same module. This is called conditional sharing in [PaPa88]. In [PaPa88], the authors first use node coloring to determine the mutual exclusiveness of the operations in the data flow graph. After that, the scheduling algorithm assigns the mutually exclusive operations into the same entry in the allocation table. Fig. 2.2 shows an example of the allocation table for the data flow graph in Fig. 2.1 where mutually exclusive operations op_2 and op_3 are assigned to the same cell. Suppose we want to calculate the switching activity of op_1 sharing a FU with op_2/op_3. The result can be obtained by slight modification of equation(3.1) as follows.

We first calculate the switching activity between op_1 and op_2. Suppose the two ordered operands for op_1 are x_1 and y_1, and the two ordered operands for op_2 operation are x_2 and y_2.

Assume that the n primary input random variables in a single initiation of the pipeline are $a_1, a_2, \ldots, a_n$ and the conditional test at the D1-J1 pair in Fig. 2.1 is $g_1(a_1, a_2, \ldots, a_n)$ while the conditional test at D2-J2 pair is $g_2(a_1, a_2, \ldots, a_n)$. Let

$$\mathcal{A} = \{(a_1, a_2, \ldots, a_n) \mid g_1(a_1, a_2, \ldots, a_n) =' False' \ and \ g_2(a_1, a_2, \ldots, a_n) =' False'\}$$

be the set containing all possible combinations of input tuples that lead to the branch where

op_2 is executed and let

$$\mathcal{E} = \{(x_1, x_2) \mid x_1 = x_1(a_1, a_2, \ldots, a_n),\ x_2 = x_2(a_1, a_2, \ldots, a_n),\ \forall(a_1, a_2, \ldots, a_n) \in \mathcal{A}\} \quad and$$
$$\mathcal{F} = \{(y_1, y_2) \mid y_1 = y_1(a_1, a_2, \ldots, a_n),\ y_2 = y_2(a_1, a_2, \ldots, a_n),\ \forall(a_1, a_2, \ldots, a_n) \in \mathcal{A}\}.$$

The switching activity between op_1,op_2 is calculated by:

$$\begin{aligned} switching(op_1, op_2) &= \sum_{(x_1,x_2)\in\mathcal{E}} f_{x_1x_2}(x_1, x_2) \times H(x_1, x_2) \\ &+ \sum_{(y_1,y_2)\in\mathcal{F}} f_{y_1y_2}(y_1, y_2) \times H(y_1, y_2) \end{aligned} \tag{B.1}$$

In practice, when we calculate the sequence of (x_1, x_2) and (y_1, y_2) values based on the applied primary input sequences, we will annotate each pair with the set they belong to (in this case, set $\mathcal{E}$ or $\mathcal{F}$). All of this calculation can be performed by a single scan through the primary input vector sequence since from the allocation table we know exactly what intermediate values (or pairs of values) must be generated and what sets must be characterized.

The switching activity between op_1 and op_3 is calculated in a similar fashion. In the power optimization phase, we treat the mutual exclusive operations as only one *conditioned* operation. The switching activity between other operation and this conditioned operation is the sum of the switching activities for each of the mutual exclusive operations. In the above example, we treat op_2/op_3 as only one operation op_{23}. Then the $switching(op_1, op_{23}) = switching(op_1, op_2)$ + $switching(op_1, op_3)$. Essentially, we merge mutually exclusive operations into an equivalent operation, then perform the permutation.

In the practice, when we construct the table for the pairs (x_1, x_2) we can use additional columns to mark each such pair with whether they belong to certain sets. At the same time, we can build the sets $\mathcal{A}$, $\mathcal{E}$ and $\mathcal{F}$ and calculate the switching activity at the same time when

we scan through the primary input vector set. From the allocation table, we know what we will need and can do all calculations by one scan.

Bibliography

[CaWo91] R. Camposanoand and W. Wolf, "High-level VLSI synthesis", PP. 256, Kluwer Academic Publishers, c1991.

[ChPo92] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R.W. Brodersen,"HYPER-LP: A System for Power Minimization Using Architectural Transformations", In Proc. of the Int'l Con. on Computer Aided Design 1992.

[ChPe95] J.-M. Chang and M. Pedram, "Low Power Register Allocation and Binding", In Proc. of the 32nd DAC, June, 1995.

[DeMi94] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw Hill 1994.

[Deng94] C. Deng, Power Analysis for CMOS/BiCMOS circuits. In Proceedings of the 1994 International Workshop on Low Power Design, pp. 3-8, April 1994.

[GaDu92] D. Gajski, N. Dutt, A. Wu, and Y.-L. Lin, "High-Level Synthesis, Introduction to Chip and System Design", Kluwer Academic Publisher, 1992.

[Gall68] R. Gallager, "Information Theory and Reliable Communication", Chap. 2, Wiley, New York, 1968.

[GeEl92] C. Gebotys and M. Elmasry, "Optimal VLSI Architectural Synthesis", pp. 148, Kluwer Academic Publishcation, pp. 148, 1992.

[GoOr94] L. Goodby, A. Orailoglu and P.M. Chau,"A High-Level Synthesis Methodology for Low Power VLSI Design". In Proc. of the IEEE Symposium on Low Power Electronics, 1994.

[Hayk91] S. Haykin, "Adaptive Filter Theory", 2nd edition, Chap5, 6, and 8, Printice Hall, 1991. Network Programming", John Wiley and Sons, 1980.

[KuAl87] Fadi Kurdahi, Alice Parker, "REAL: A Program for Register Allocation", In Proceeding of 24th Design Automation Conference, June 1987.

[LaRa94] P. Landman and J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis", International Workshop Low Power workshop pp. 165-170, Napa Valley, CA, April 1994.

[Leng90] T.L. Lengauer, "Combinatorial Algorithms for Integrated Circuit Layout", John Wiley and Sons, 1990.

[MeRa94] R. Mehra and J. Rabaey. "Behavioral Level Power Estimation and Exploration.", In proceeding of the 1994 International Workshop on Low Power Design, pp. 197-202, Apr, 1994.

[BuNa93] R. Burch, F. Najm, P. Yang and T. Trick, "A Monte Carlo approach for power estimation". In IEEE Trans. on VLSI page 63-71, March 1993.

[PaPa88] N. Park, A. Parker,"Sehwa: A Software Package for Synthesis of Pipelines from behavioral Specifications", In IEEE Trans. on CAD, vol 7, no. 3, pp. 356-371, March 1988.

[Papo91] Athanasios Papoulis, Probability, Random Variables, and Stochastic Processes. Third Edition, section 6.3.

[PoCh91] R. Powell and M. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips", In IEEE Trans. on Circuits and Systems, Vol 38, No. 6, pp. 646-650, June 1991.

[RaJh94] A. Raghunathan and N. Jha, "Behavioral Synthesis for Low Power", In Proc. Int'l Conf. on Computer Design 1994.

[Stok91] L. Stok, "Architectural Synthesis and Optimization of Digital Systems", Ph.D Dissertation, Eindhoven University of Technology, 1991.

[SvLi94] C. Svensson and D. Liu, "A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips". In Proc. of the 1994 International Workshop on Low Power Design, pp. 171-176, Apr 1994. of Data Paths in Digital Systems," IEEE Trans. on CAD, vol CAD-5, no.3, pp. 379-395, July, 1986.

[ViOm79] A. Viterbi and J. Omura, "Principle of Digital Communication and Coding", McGraw-Hill, New York, 1979.