A Design Methodology for Behavioral Level Power Exploration :- Implementation and Experiments

Hartej Singh
Daniel D. Gajski

Technical Report #97-28
July 9, 1997

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(714) 824-7063

hsingh@ics.uci.edu
gajski@ics.uci.edu

Abstract

This work describes an integrated design methodology focusing on power exploration at behavioral level. We present a behavioral-level tool for architecture selection and power estimation within the SpecSyn [GVNG94] environment. The implementation has two major steps of scheduling and power estimation. Power analysis is done in both steps; it is approximate in the former and detailed in the latter. A combination of empirical and analytical approaches is used for power estimation. We provide several case studies of common benchmarks to illustrate the design methodology as also its interactive and efficient nature that enables fuller design space exploration. We evaluate power versus performance trade-offs for different implementations of each example, using our tool. An important premise for this work is accurate characterisation of resource library for power.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Related Work 2

1.3 Targets 2

1.4 Organization 3

2 Proposed Methodology **3**

2.1 Architecture Selection 4

2.2 Power Estimation 5

3 Design Methodology **6**

3.1 Scheduling and Allocation 6

3.1.1 Resource-constrained Scheduling 7

3.1.1.1 Schedule Generation 7

3.1.1.2 Analysis 7

3.1.1.3 Allocation Refinement 7

3.1.1.4 SRA Example 9

3.1.2 Time-constrained Scheduling 10

3.1.2.1 Allocation 10

3.1.2.2 Analysis 11

3.1.2.3 Schedule Refinement 11

3.1.2.4 HAL Example 11

3.2 Power Estimation and Trade-offs 13

3.2.1 Profiling and Probability Analysis 14

3.2.2 Switched Capacitance Estimation 14

3.2.3 Supply Voltage Assignment 15

3.2.4 ELL Example 15

4 Implementation **16**

4.1 Allocation and Schedule Generation 16

4.1.1 Component Library 16

4.1.2 Scheduling 17

4.2 Analysis and Refinement 17

4.2.1 Probability Analysis and Access Frequency 17

4.2.2 Power Computation from Capacitance Models 18

4.2.3 Allocation Refinement 18

4.3 Limitations of Delay and Area Estimation 19

5 Experimental Results **19**

5.1 Experiments in Architecture Selection 20

5.1.1 HAL Example 20

5.1.2 Elliptic Wave Filter 21

5.1.3 FIR Filter 22

5.1.4 Kalman Filter 22

5.1.5 Jacobian Filter 23

5.2 Experiments in Power Estimation 24

5.2.1 DCT: Sequential Version 26

5.2.2 DCT: Optimized Versions (no pipelining) 26

5.2.2.1 Loop Unrolling 26

5.2.2.2 Loop Unrolling and Chaining 27

5.2.2.3 Loop Unrolling and Multicycling 27

5.2.2.4 Loop Unrolling, Chaining and Multicycling 28

5.2.3 DCT: Pipelined Versions 28

5.2.3.1 Loop Pipelining 29

5.2.3.2 Loop Pipelining with Pipelined Multiplier 29

5.2.4 DCT: Result Analysis 30

6 Conclusions **31**

References **32**

Appendix **33**

A SRA Algorithm Description **34**

B HAL Specification and DFG **34**

C Component Library **35**

D ASM Charts for DCT **38**

List of Figures

1 Global Picture . . . 3
2 Proposed Methodology . . . 4
3 Architecture Selection . . . 6
4 SRA Example . . . 9
5 HAL Example . . . 12
6 Power Estimation . . . 14
7 ELL Example . . . 15
8 Program Flow . . . 17
9 ASM Chart for Sequential Version of DCT . . . 25
10 Comparison of Sequential, Loop Unrolled and Optimized DCT Versions . . . 30
11 Comparison of Optimized and Loop Pipelined Versions of DCT . . . 30
12 Square Root Algorithm Flowchart . . . 34
13 HAL: DFG and Textual Description . . . 35
14 Matrix Multiplication with Loop Unrolling . . . 38
15 Matrix Multiplication with Loop Unrolling and Chaining . . . 39
16 Matrix Multiplication with Multicycling and Loop Unrolling . . . 40
17 Loop Pipelined Matrix Multiplication . . . 41

List of Tables

1 Evaluation of HAL Example . . . 21
2 Elliptic Wave Filter . . . 21
3 FIR Filter . . . 22
4 Kalman Filter . . . 23
5 Jacobian Filter . . . 23
6 DCT: Sequential Version . . . 26
7 DCT: Loop Unrolling . . . 27
8 DCT: Loop Unrolling and Chaining . . . 27
9 DCT: Loop Unrolling and Multicycling . . . 28
10 DCT: Loop Unrolling, Chaining and Multicycling . . . 28
11 DCT: Loop Pipelining . . . 29
12 DCT: Loop Pipelining with Pipelined Multiplier . . . 30
13 Component Library for Examples . . . 36
14 Delay vs. Supply Voltage . . . 36
15 Component Library for Implementation . . . 37

1 Introduction

Recent advances in VLSI technology allow fabrication of chips containing over one million transistors. At the same time, several application fields are opening up, for example, multimedia or interactive video applications, that require ever more complex design due to increased functionality and high performance demands. As chip complexity increases, designers need to initiate the design process at higher levels of abstraction rather than gate or logic level. This is because functionality and trade-offs are better understood at behavioral or system levels. In this context, behavioral synthesis is an important element of the design process.

Behavioral synthesis is a process of synthesizing a design from a given behavioral description to a register-transfer level (RTL) description. Behavioral descriptions are usually programs, algorithms, flowcharts, dataflow graphs etc. The RTL structure refers to a set of inter-connected components, along with a netlist, specifying the connectivity. The components may be functional units (ALU's or multipliers), storage units (registers and memories) or interconnection units (muxes or buses).

In general, behavioral synthesis consists of three major tasks: allocation, scheduling and binding [GWDL94]. **Allocation** refers to specifying the number and type of resources needed for the design. **Scheduling** is the task of partitioning the behavior into different control steps or states, such that each step is executed in one clock cycle. **Binding** deals with assignment of each variable to a storage unit, each operation to a functional unit and each data transfer to interconnect units. These three tasks are closely related to each other.

Given the complex nature of design evaluation using behavioral synthesis, there is need for an integrated methodology that provides minimum design time and optimizes the required metrics. Next, we discuss the desirable features of deisgn methodology, the relevance of power optimization for it, and related research work done in the past.

1.1 Motivation

Conventionally, behavioral synthesis algorithms have targeted delay and area as optimization metrics. However emerging applications have thrust low-power oriented design methodologies to the fore-front. Due to reasons of portability, battery life, reliability and packaging cost, optimization for power is really important for any reasonably complex design. Power optimization approaches at the behavioral level are significant since research results indicate that higher levels of abstraction have greater potential for power reductions.

The three design quality metrics of power, area and delay, make the design space three-dimensional. These metrics have complex inter-relationships that are design-specific. For a given behavioral specification, there are many implementation choices. Also, different applications typically have different constraints for the metrics of power, area and delay. A designer needs to select an op-

timum configuration for some specification, with regard to given constraints. An integrated framework is required for the designer to visualize the overall relationship as well as characterize completely some point in the design space.

For effective exploration, there are three main requirements. These are:

Interactive synthesis: Control of design flow by designer is needed for practical designs, since designer experience and intuition is indispensable.

Efficient evaluation: To enable complete exploration, each point in design space should be evaluated rapidly.

Characterization of design space: An effective means to represent metric relationships in design space is required. This is because these relationships are usually quite complex, and sometimes the larger perspective is lost.

Keeping the above points in perspective, we propose a design methodology that provides an integrated approach to behavioral synthesis for low power. It incorporates the above features as well as optimization techniques for power reductions. Different tradeoff options can be explored by the designer. However, it would be useful to review some of the relevant research work done earlier in this sphere.

1.2 Related Work

There has been a lot of research in the area of power estimation and optimization for power. An overview of power estimation techniques in VLSI circuits is given by Najm [Naj87]. Traditionally, circuit level simulators such as SPICE have been used for power estimation. Since transistor level simulation is too slow for use with large designs, logic level simulation was proposed (Irsim). Even this does not allow fast exploration. An architectural level power estimator based on profiling and state trace is described in [ZAG97]. Landman [LR93] proposes a modeling approach for module characterization. Mehra and Rabaey [MR94] address the problem of estimating power from a behavioral description using a combination of analytical and stochastic techniques. A profile driven approach is presented for low power behavioral synthesis by Kumar *et al* [KKRV96]. Raghunathan *et al*,[RJ94] and [SR96], have developed a tool framework for low power behavioral synthesis, using scheduling and allocation based on a compatibility graph.

However, other than estimation, most of these approaches do not accomodate design space exploration. Even though an integrated CAD environment for low power design was proposed in [LMR95], it has several limitations. It is vague with respect to actual behavioral synthesis of the designs and does not specify the level of interaction or automation, even though it does employ a modeling approach at different levels of abstraction.

1.3 Targets

The targeted aims for this project were as follows:

- Develop a fast and relatively accurate methodology for behavioral synthesis

that provides optimization for power.

- The environment for design should be interactive as well as iterative.
- The important constituents of the design process such as allocation, component library, clock cycle, pipelined components, etc., should be easily modifiable by the user.
- Important design information such as number of memory accesses, register loads, bus loads and functional unit usage should be provided to the user, as well as be a parameter for automatic refinement of the design.

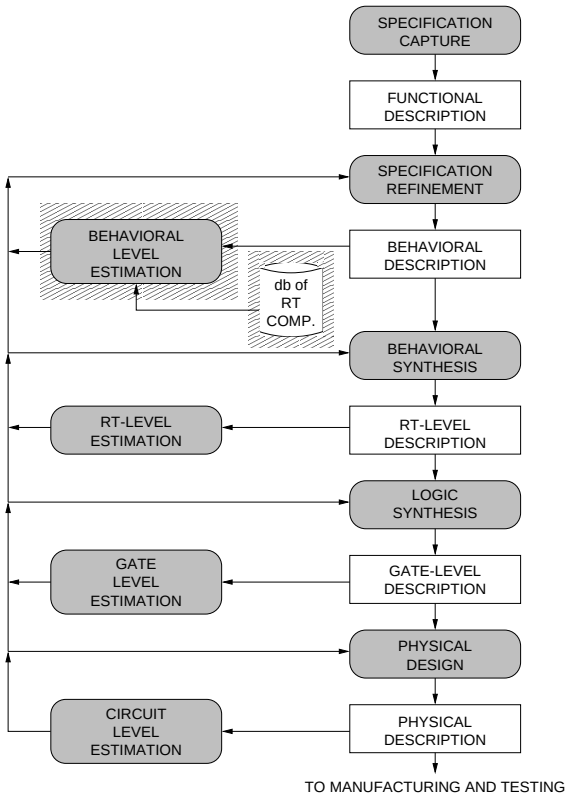

Figure 1: Global Picture

Figure 1 shows the global perspective as to where this work would fit in the ASIC design process. The areas covered by this work are shown hatched. For a typical ASIC top-down design process, the lower levels are associated with increasing costs for design exploration. Hence, it is useful to have an estimation tool at higher levels of abstraction, for ensuring rapid exploration and iterative refinement. In an earlier work by Agrawal [Agr96], a library of RT components characterized for power was generated using the estimation tool, PRESTO. We aim to build upon the results from that project in order to move the estimation of power one level up from RT level to the behavioral level.

1.4 Organization

The second section provides a brief introduction to the proposed design methodology. In the third section, a detailed description is provided for the steps involved, with illustrative examples. The fourth section deals with implementation details while experimental results are discussed extensively in the fifth section. The sixth section summarises the conclusions from the experiments, reviews the goals achieved and discusses areas for future work.

2 Proposed Methodology

The target is to enable the designer to select an optimal low power design, starting from a given behavioral specification. As illustrated in Figure 2, there are two important steps in the design process. An architecture imple-

mentation is chosen in the first step. For the second step, the chosen architecture implementation is evaluated for power dissipation vs. delay trade-offs.

The first step uses allocation techniques. A schedule is prepared for the design. An approximate evaluation of the scheduled design is performed for power and performance. The user constraints on power, delay and area may not be satisfied the first time, hence this step may iterate several times.

In the second step, the motivation is exploration of design space for power. So performance constraints are varied through a range of values and corresponding power dissipation estimated for specified allocation and schedule. In contrast to the first step, the power estimation method used is more comprehensive, since detailed profiling information is incorporated at this stage. Also, the controller and interconnect models may also be used to provide more realistic estimation. This step repeats each time the performance constraint is violated.

When the user constraints are flexible, the designer may choose to go in for further exploration. The exploration is enabled by repeating the design process after modifying the constraint values. Thus the design process is essentially iterative in nature, to accommodate constraint variations and enable complete exploration. A brief outline of the main steps involved is given below.

2.1 Architecture Selection

Allocation and scheduling, followed by binding are performed for the CDFG representation of the specification. The designer has two options for scheduling: resource-constrained or time-constrained. For either approach, a schedule and allocation are specified using applicable algorithms. Next, a quick analysis of this schedule is performed for power dissipation, area and delay. Since this analysis has to be rapid, it is neither very detailed nor comprehensive, but aims to have a quick approximate value for the metrics. We use the concept of relative accuracy or consistency rather than absolute accuracy for this analysis. A refinement step follows the analysis, which tries to modify the allocation or schedule to minimize the constraint viola-

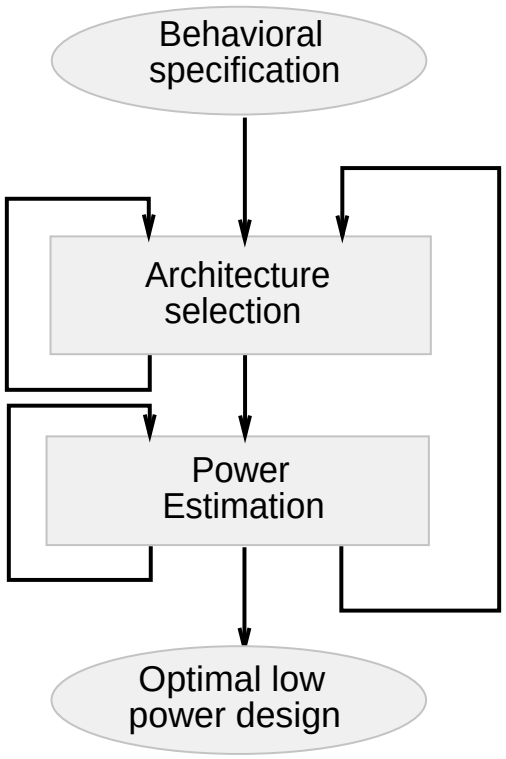

Figure 2: Proposed Methodology

tions. Several iterations may be needed before reaching a satisfactory architectural implementation.

2.2 Power Estimation

For a given circuit, the power dissipation is computed as

$$P = 0.5 * V_{dd}^2 * C * f \quad (1)$$

where V_{dd} is supply voltage, C is the average switched capacitance and f is the switching frequency [Agr96].

The basic steps involved in power estimation are:

- *Estimation of switching activity*: The switching activity is estimated based on number of accesses for each functional and storage unit. The accesses can be determined from the profile and state trace information related to the CDFG.

- *Relating switching activity to switched capacitance*: The average switched capacitance per access for each unit is known from the component library. Combining this information with the total accesses for each unit, the switched capacitance is calculated. The capacitance switched in the interconnect can be approximated through an appropriate model.

- *Assignment of supply voltage for the specified execution time* : For a specific execution time, delay vs. supply voltage curves for the functional units are used to generate a minimum value of the supply voltage. The execution time value is varied over a range and corresponding supply voltage values selected. This iteration forms the basis for analysis of power-performance trade-offs.

Relative Accuracy: The concept of relative accuracy is significant for behavioral level power estimation. Since at the high level, many low-level implementation details are not available, we need to be sure that estimates at this level bear some correlation to final design values. It is here that the concept of relative accuracy comes in useful. This concept implies that the power dissipation of a particular specification is closely related to the operational activity present in the CDFG for the design [Lan96]. If this is the case, we can compare two different specifications on the basis of high level estimates for power, with sufficient confidence that the final estimates for each of them will be related in a similar manner to each other.

With reference to this concept, Mehra and Rabaey [MR94], provide several examples of behavioral-level power estimation that show good correlation to final implementation values. Besides operational activity (functional unit accesses), they also use some sort of heuristical analysis for storage accesses as well as bus accesses. From comparison of their results with an architectural (register-transfer) level power estimator, SPA [LR95], they get a good correlation between high level estimates and architectural level calculations. Based on this, we follow a similar approach for high level power analysis, i.e.,

try to model the activity contained in a particular CDFG, and then relate it to actual switched capacitance using capacitance models for the resources.

Also, it might be pointed out that the intrinsic activity for any particular CDFG remains the same. The potential for power optimizations arises through: (a) use of resources with different switched capacitance values, (b) varying the schedule (by using multiple units), (c) pipelining some of the constraining operations, such as multiplications, or (d) by minimizing slack of the resources. If the schedule is varied to perform the same CDFG specification in less number of states, the supply voltage might be lowered while keeping constant execution time. This has a significant effect on lowering power dissipation.

3 Design Methodology

This section discusses the detailed features of the design methodology. The two basic steps of scheduling and power evaluation, are composed of many sub-steps, with several options available to the designer. The interactive nature, fast design loop and representation of trade-offs will be illustrated in the following descriptions.

3.1 Scheduling and Allocation

This step deals with obtaining an architectural implementation from the behavioral specification that would be optimized for low power dissipation within specified constraints. An allocation and schedule are specified for the CDFG using either time-constrained or resource-constrained scheduling. These are subjected to a quick analysis for power, area and delay. If the constraints on these metrics are violated, the allocation and schedule are modified and again analyzed. Since the scheduling, allocation and refinement algorithms differ for the resource-constrained approach as compared to time-constrained approach, they are discussed in separate subsections. Figure 3 illustrates the steps involved for both approaches.

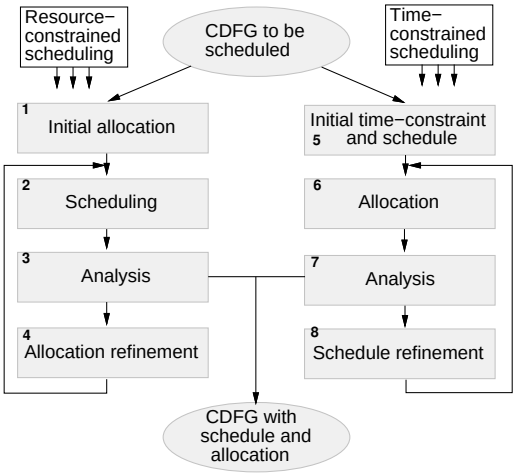

Figure 3: Architecture Selection

Limitation of schedule: One limitation of the schedule is that delay calculations are simplified to include only the functional unit delay. This is not strictly true in practice, as finite delays are associated with register accesses, bus drivers and the interconnect wiring. However, in all subsequent examples, we consider only the maximum functional unit delay as constraining the clock period. For illustrative examples in sub-sections 3.1.1.4 and 3.1.2.4, we also assume the clock

cycle to be equal to the maximum operational delay.

3.1.1 Resource-constrained Scheduling

In Figure 3, the boxes 1 through 4 depict the sequence of steps involved for this step. The tasks are to select an allocation, generate a schedule, perform an analysis of the schedule followed by allocation refinement to satisfy constraints. Each time the allocation is modified, the schedule is generated again and analyzed. When the constraints are satisfied, this step terminates.

An initial allocation for the CDFG is specified by the designer or optionally by a heuristical analysis of the CDFG (box 1).

3.1.1.1 Schedule Generation Once the resources for the CDFG have been allocated, it is scheduled by an appropriate algorithm such as list scheduling (box 2). In fact, any other algorithm of the resource-constrained scheduling category [GWDL94] might be used here. The goal is that schedule generation should be efficient.

3.1.1.2 Analysis A quick analysis of the scheduled CDFG is then performed (box 3). The objective is to get approximate estimates for power dissipation, chip area, performance and minimum state slack (for each functional unit). The difficult part is to estimate the power dissipation, the other metrics are generally easier to compute.

Power analysis needs to take into account the number of register and memory accesses, number of functional unit (F.U.) operations, number of bus loads and bitwidths of resources used. This information is combined with power characterization models (for resources) to estimate power dissipation. At this step, models of interconnect as well as controller for power dissipation may not be used. Relative accuracy rather than absolute accuracy is more important here, since a detailed evaluation is again done in power estimation step. Approximate models may be used to facilitate quick analysis.

The concluding part of analysis compares values obtained for average power dissipation, chip area and performance with initial constraints. For each metric, if the value is within some margin of constraint, it is assumed to satisfy it. In case the value is sufficiently larger, this is a violation of constraint. But if the value is much smaller than the specified value of constraint, there is slack in that constraint.

3.1.1.3 Allocation Refinement The task here is to reduce the constraint violations or slack detected in the analysis step by making refinement changes to the allocation, while trying to attain minimum power dissipation (box 4).

There are two types of refinement changes: optimizations and trade-offs. Optimization changes are those which have no associated cost in terms of some other metric. Trade-off changes imply that an improvement for some metric is made at the cost of another metric. For example, power may be reduced by using smaller units, however smaller units will gen-

erally have larger delay. However, if available slack allows use of the smaller unit, the performance of the schedule will not be affected. This is termed as optimization. But if there is no slack, using the smaller unit will lower the performance. In the design flow, optimization approaches precede trade-off strategies, since they have no associated cost. However, refinement allows designer interaction for making decisions regarding these optmizations or trade-offs.

Refinement changes will typically be accompanied by change in constraint violation or slack. So, an important aspect of allocation refinement step is keeping track of modifications to constraint violations or slack by the changes.This will enable in terminating the refinement process once the constraint violations have been reduced to permissible levels. The task of evaluating effect of refinement on violations/slack is simple in the case of area related changes. However in case of power violation, information about unit accesses is required. Again, for performance violations, it may sometimes be difficult to reflect refinement changes in terms of modification to constraint, especially since the schedule may also change.

The descriptions that follow illustrate paradigms for allocation refinement when constraints are violated. Similar paradigms (not discussed here) would be applicable for the case of slack in constraints. At the completion of these steps, it may be necessary to reschedule and analyze for constraint violations.

Power constraint: Violations in power constraint imply that power hungry units need to be replaced. The optimization approach tries to replace functional units with lower power units, such that available state slack is not violated. Once the optimization potential is finished, and constraint is still not satisfied, trade-off strategies are followed.

For trade-offs, the average power dissipation of a resource (per access) per bit, frequency of accesses, bitwidth and unit multiplicity is used to create a list of resources sorted by power dissipation. The units responsible for greater power dissipation are replaced by lower power and/or slower units, if available in the resource library. Also, heavily utilized units are replicated so that number of states in the schedule may be reduced.

Performance constraint: The violations for performance can be reduced by substituting slower units with faster ones or replicating heavily utilized units. Every change may be associated with higher area or higher power costs. It may be important to *not* make unbalanced trade-offs for power. For example, if the performance gain offered by a substitution is marginal but the power dissipation is greatly increased, the trade-off is not viable. Incidentally, this may not be quite obvious.

Area constraint: For violations of the constraint for area, we need to use smaller units or decrease the multiplicity of units. First, we carry out optimizations, if possible. This would refer to making substitutions of larger units by smaller ones within the available slack, considering smaller units will generally be slower. For trade-offs, such substitutions are carried out, even if available slack

is violated. However, for reducing the multiplicity of units, unit utilization would be the criterion. An instance of an unit would be removed only if utilization is below some threshold value.

3.1.1.4 SRA Example Next, we consider a simple example to illustrate the above steps. This example is designed to compute the square-root approximation (SRA) of two signed integers, a and b, by the following formula:

$$\sqrt{a^2 + b^2} = max((0.875x + 0.5y), x) \quad (2)$$

where

$$x = max(|a|, |b|), y = min(|a|, |b|) \quad (3)$$

A flowchart description for design steps of scheduling, analysis and allocation refinement is provided in Figure 4. The specification details from [Gaj95] are provided in Appendix A and the relevant component library is described in Appendix C. For purposes of analysis, functional unit delay is assumed to be related to supply voltage according to Table 14.

For this example, the constraints are specific only in that power reduction has priority over performance as well as area considerations. Since we begin with a minimum allocation, we may be able to improve performance, on subsequent iterations. However, for illustration purposes, we shall select the execution time of the first iteration as the available execution time for all subsequent iterations. *The area cost is secondary to power, so we allow*

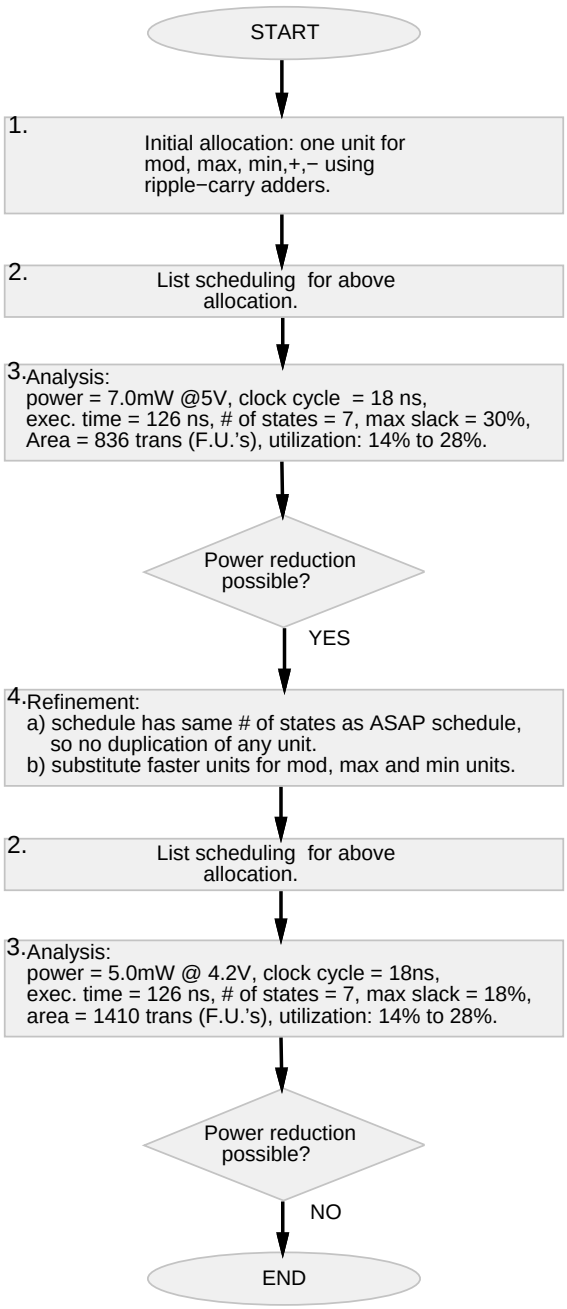

Figure 4: SRA Example

increase of area, as long as it leads to some power reduction.

The CDFG for SRA is scheduled using ASAP algorithm, to find the length of critical path, which is six states. The initial allocation is fixed, based on types of operations (box 1). The CDFG is scheduled for current allocation using list scheduling (box 2). The next step is analysis (box 3). Power is estimated based on frequency of accesses for functional units, number of loads for registers and buses and average switched capacitance per access. Power dissipated is 7.0 mW @ 5V supply voltage. The minimum clock period is 18 ns and number of states is seven. Area cost for functional units is approx. 830 transistors.

User constraints aim at power reduction, while having same performance with possible area increases. So after analysis, in the refinement step (box 4), there are two considerations. One is to check if any unit multiplicity can be useful. The other is to use faster components for higher delay operations. This means that if the units with greater delay are replaced by faster units, the state delay will decrease. Or, for the same performance, a lower supply voltage may be assigned for lower power, even though faster units have higher power dissipation than slower units.

Since present schedule has only one state more than ASAP schedule, therefore any multiplicity will not be very useful. But for operations with large delays (mod, max and min), the library has faster versions (using carry look-ahead adders in place of ripple carry adders). So mod, max and min units are replaced by faster versions.

Again, the schedule is prepared (box 2). It is same as the previous one, but analysis (box 3) yields lower power value of 5.0 mW @ 4.2V (decrease of 30% in power) for same clock period of 18 ns and same execution time. The area cost increases to around 1400 transistors for functional units. It may be remarked that the faster units can function at lower supply voltage of 4.2 V (from Table 14) while still satisfying the clock cycle of 18ns. For given component library, there is no further scope for refinement, hence the scheduling process concludes.

This simple example demonstrates the sequence and motivation of steps for power optimization.

3.1.2 Time-constrained Scheduling

The steps involved in this approach are assigning the time constraint, generating a schedule, specifying allocation for the generated schedule, analyzing the scheduled CDFG and carrying out schedule refinements for meeting constraints. These are depicted in boxes 5 through 8 of Figure 3.

The time constraint is specified by the designer in terms of number of states and minimum cycle time. Next, a schedule is generated by using heuristics such as the force-directed scheduling algorithm (box 5) as described in [GWDL94]. Once the schedule is ready, the allocation is performed, so as to satisfy the schedule.

3.1.2.1 Allocation The resource library is referred for specifying the allocation (box

6). If relative priority is assigned to the constraints, automatic allocation may be good enough. Otherwise, the designer is able to vary the choice of allocated units with regard to constraints. For example there might be different multiplier versions in the resource library. If power constraint has a higher priority than performance, a low power but slower version may be chosen. Similarly, the designer would be able to make changes to allocation to account for exceptions to given constraint priority.

3.1.2.2 Analysis This step (box 7) is identical to quick analysis step described for resource-constrained scheduling case. The same considerations and descriptions apply here, too.

3.1.2.3 Schedule Refinement The step (box 8) is analogous to allocation refinement step for resource-constrained scheduling. The objective is same: to reduce constraint violations/slack by making appropriate changes. However, the difference is that changes are made directly to the schedule by varying the number of states. The possible modifications for a schedule are much limited than for allocation. The constraint on number of states can be either increased or decreased. This makes schedule refinement much more simpler in concept as well as implementation, as compared to allocation refinement. However, the effects are related to actual changes in more complex fashion. The granularity of modification effects may be larger. It may also be difficult to anticipate accurately the effect of schedule changes on area and power constraints.

An important point to note is that designer input for schedule refinement decisions is more relevant and necessary, for each change may have unpredictable effects. This is different than for allocation refinement, where automation can produce good results. The designer's experience and intuition can be used here to control the optimizations and tradeoffs.

Power constraint: In case of violation of power constraint, the number of states needs to be reduced, so that for given execution time, the state delay may be increased. This in turn, would enable a lower supply voltage for the units, lowering the power dissipation.

Performance constraint: Similar to above, the two options available are to change the constraint on number of states or vary the minimum cycle time value. For violations of performance requirements, the number of states needs to be decreased and/or the cycle time should be reduced.

Area constraint: If the area constraint is violated, then the number of states needs to be increased. This may result in lower multiple operations of same type, and reduce the multiple instances of units. Alternatively, the minimum cycle time may be increased so that smaller units can substitute for larger ones. The opposite approach can be taken if there is slack in the area constraint.

3.1.2.4 HAL Example The HAL second order differential equation solver implementation is discussed to illustrate this ap-

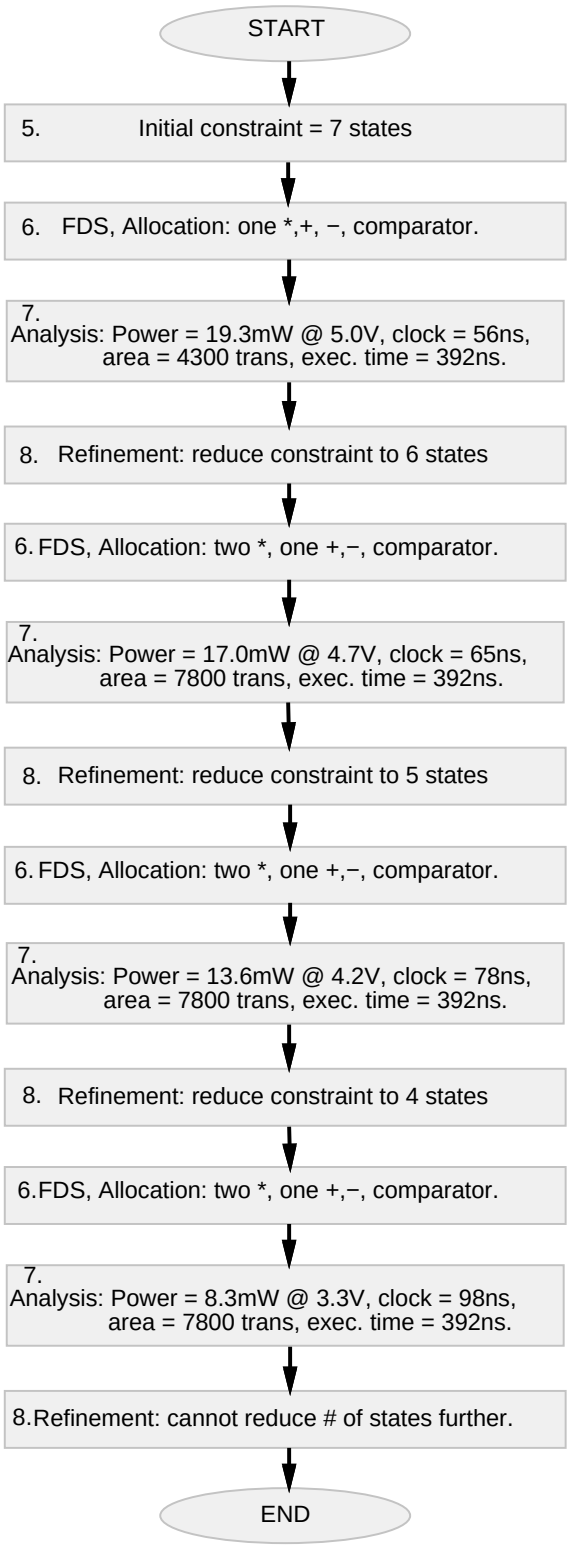

Figure 5: HAL Example

proach. The specification as well as DFG and component library for this example are given in Appendix B and Appendix C. The functional unit delay is assumed to be related to supply voltage as given in Table 14. Figure 5 shows the iterative steps of scheduling, analysis and refinement. *For this example, the constraints are the same as for the SRA example: power reduction has priority over performance as well as area considerations.* Since we begin with a minimum allocation, we may be able to improve performance, on subsequent iterations. However, for illustration purposes, we shall select the execution time of the first iteration as the available execution time for all subsequent iterations.

The first step is to find the ASAP schedule, which in this case has four states. Next, an initial constraint of seven states is specified (box 5). Using force-directed scheduling, the schedule is specified for minimum allocation (box 6) which, in this case, is one unit each of a multiplier, adder, subtractor and comparator. The schedule is analyzed (box 7) in same manner as the SRA example, giving power dissipation as 19.3 mW @ 5.0V supply voltage. Clock period is 56 ns and area for functional units is 4300 transistors.

For refinement (box 8), the aim is to reduce power. This implies the number of states be reduced. So the constraint is set at six states. Again the schedule and corresponding allocation are analyzed (box 6). The schedule now requires two multipliers as compared to one for earlier schedule. Power dissipation for same execution time (392 ns) is 17.0 mW @ 4.7V, however area for functional units in-

creases to 7800 transistors (box 7).

Next, the constraint is reduced to five states (box 8). Allocation remains the same although schedule changes (box 6). Power dissipation for same execution time is 13.6 mW @ 4.2V and there is no increase in area (box 7).

When the constraint is further reduced to four states (box 8), power decreases to 8.3 mW @ 3.3V for same allocation (box 6 and 7) . Since ASAP schedule also has four states, optimization and refinement process concludes (box 8).

For this example, final power decreased by nearly 60% from the initial value. Area required is less than area for ASAP allocation, while performance is same.

3.2 Power Estimation and Trade-offs

General considerations: Since power estimation is done at the architectural (RTL) level, the gate, circuit and layout level details have not been specified. A floorplan is also not available, making analysis of interconnect and clock distribution networks difficult. Therefore, some approximation is inherent in the estimation.

Estimation methods: These may be analytical or empirical. Analytical techniques relate the power dissipation to fundamental quantities that describe the physical capacitance and inherent switching activity of a design [Lan96]. But since these methods use very little information about inputs, power predictions based on them have little relation to actual hardware. On the other hand, empirical methods employ a macromodeling approach. The library of resources is characterized for power dissipation, which implies that the power dissipations of resources for different supply voltages, under some assumptions for input data, are pre-computed. There may be multiple components having same functionality but different power, delay and area costs. Designs that have components from the library can be modeled for power using the characterizations. Usually, these empirical models have a strong link to actual implementations. For the proposed methodology, a combination of analytical and empirical approaches is adopted based on Zhu [ZAG97] and Agrawal [Agr96].

Effect of supply voltage: An important aspect of power dissipation is that it is proportional to the square of the supply voltage. This implies it is desirable to use minimum value of supply voltage. But functional unit delay is a function of supply voltage. Thus if available delay changes, the minimum supply voltage required is different. This relationship is the basis of power performance trade-offs. By incrementally varying the execution time and finding corresponding minimum supply voltage (or vice versa), the trade-offs are evaluated. Figure 6 depicts the salient features of power estimation and trade-off process.

Power dissipation: As stated above, in order to estimate power, a combination of empirical and analytical approaches is taken. A dynamic activity analysis of the architecture is done for typical input vectors. This is performed by profiling the CDFG and generating

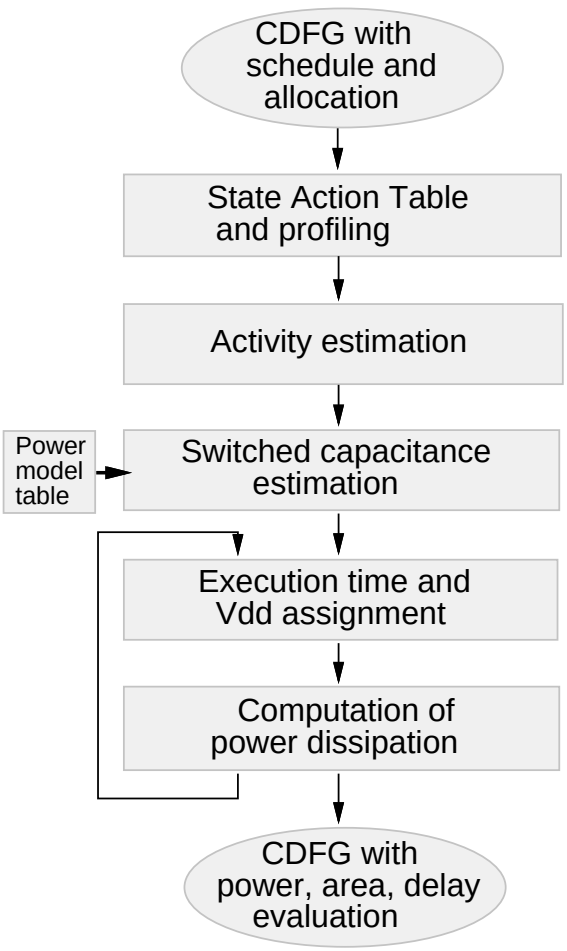

Figure 6: Power Estimation

a state trace [ZAG97]. Capacitance models of the resources are used to find the switched capacitance for the entire schedule and allocation.

Once the switched capacitance is known, the other variables are supply voltage and frequency of operation. If the execution time is fixed, frequency or clock period can be found from number of states in the schedule. This value of clock period is used to find the minimum value of supply voltage. Then power dissipation can be calculated. Alternatively, for each of a given set of supply voltages, the execution time (and hence, power dissipation) may be computed.

3.2.1 Profiling and Probability Analysis

There are two approaches; the ideal one is that for each behavioral specification, we generate the profiling information by conducting a functional simulation. The other approach is simple and basic. We assume all branches at a forking point to have equal probability of execution. Since the first approach is complex and requires input by the user, we are not implementing it currently. However, it is possible for the user to assign probabilities to each node of the CDFG, corresponding to its execution frequency from profile information. If the user does not supply this information, we follow the simple approach.

3.2.2 Switched Capacitance Estimation

The library resources are characterized for power using white noise inputs [Agr96]. For example, the capacitance for accessing a register of **N** bits is given. From activity and probability analysis, the probable number of times a register is accessed is known. Thus the total capacitance switched for different register accesses can be easily calculated. Similarly for functional units, memories, etc. the equivalent switched capacitance is computed. The components of architecture are

divided into datapath, controller, memory and interconnect.

Datapath and controller: The activity vector technique [ZAG97] coupled with look-up for capacitance values for the datapath resources [Agr96] generates values that are reasonably accurate. The controller is divided into four parts namely the state register, decoder, next state logic output logic for modeling for power [ZAG97].

Memory: The capacitance switched is related to number of memory accesses, word size, address space and specific memory capacitance.

Interconnect: The capacitance switched can be based on empirical models. For the ELL example, we use a derivative of Rent's rule [ZAG97].

3.2.3 Supply Voltage Assignment

As discussed above, the target is to use a minimum supply voltage. The supply voltage vs. delay curves for the functional units are referred to decide the minimum value possible for the specified execution time. Alternatively, for specified set of supply voltages, the execution time is computed. Units with large delays and units on the critical path constrain the value to be chosen. The assumption is that all units for a particular behavioral specification are powered using a single supply voltage.

3.2.4 ELL Example

We discuss the fifth order elliptic wave filter (ELL) example to illustrate the power esti-

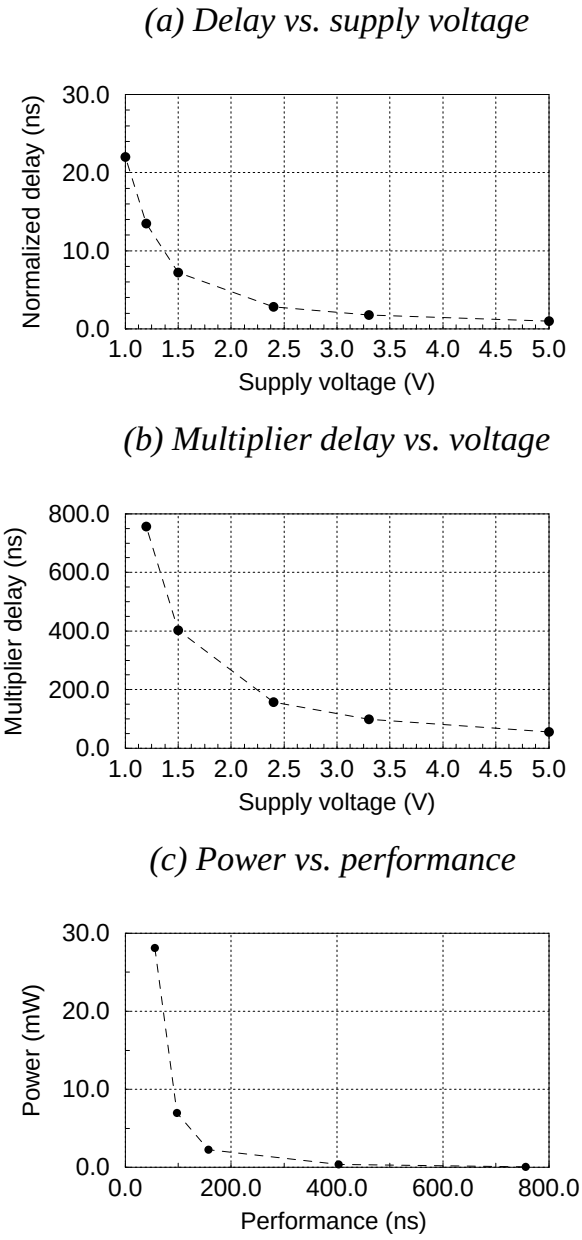

Figure 7: ELL Example

mation and trade-off procedure. The allocation and schedule has been specified, there are 20 states in the schedule with 2 multipliers and two adders.

From profile and state action table, activity vector technique of Zhu et al [ZAG97] is used to estimate the switching activity. Using power characterization models from Agrawal [Agr96], switching activity is translated to actual switched capacitance. For datapath, this value is 1046.9 pF. For controller, it is 336.8 pF (values are quoted directly from Zhu et al [ZAG97]).

Next, we compute execution time for supply voltages of 1.2, 1.5, 2.4, 3.3 and 5.0 volts, given that clock period is equal to multiplier delay. Multiplier delays are figured by scaling normalized delay-supply voltage curve (refer Appendix C, Table 14). The resulting delays or performance (in terms of clock period, number of states is 20 in all cases) values are used to compute the power dissipation. The power-performance curve depicts this relationship in Figure 7.

4 Implementation

The parts corresponding to the resource-constrained scheduling approach for architecture selection have been implemented in the SpecSyn environment. SpecSyn [GVNG94] is a high-level synthesis tool developed at CAD-LAB, University of California, Irvine. The current implementation corresponds to the architecture selection paradigm for resource-constrained scheduling. In the following sections, an overview of the program structure is provided (refer Figure 8).

4.1 Allocation and Schedule Generation

The allocation is flexible in that it is specified by the user. A text file in a specific format, serves as the allocation file. This allocation file can even be generated from a default specification, if the user does not specify it. It contains the details of each functional unit present in the ASIC chip, such as type, operations performed, delay, area, average capacitance switched per access, bitwidth, number of control signals, and whether the unit is pipelined or not. The allocation file is based upon a component library that is described next.

4.1.1 Component Library

This has two versions of units, one is high speed, high power version, and the other is slower but low power version. Units from both these versions are used interchangeably for different architectures. There are some units like multipliers, for which pipelined versions are also available, for both slower and faster libraries. It is specified in the same format as the allocation file and it is possible for the user to make custom modifications, additions or other changes to it very conveniently. Characteristics of library components are provided in Table 15 in Appendix C.

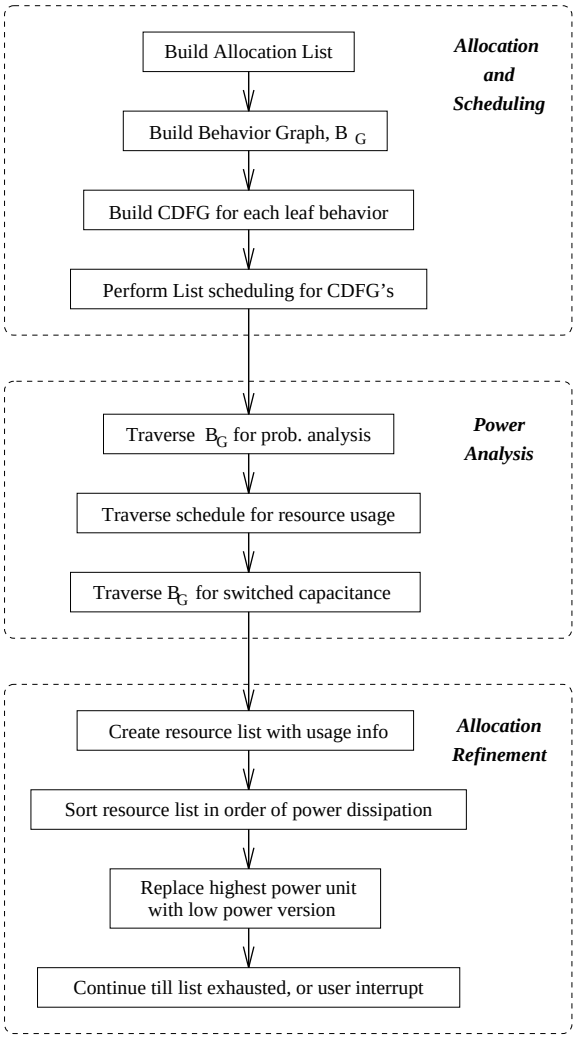

Figure 8: Program Flow

4.1.2 Scheduling

As mentioned earlier, scheduling is based on a list scheduling algorithm [GWDL94]. This algorithm takes the behavioral specification in SpecCharts language as input and constructs a Control Data Flow graph. This CDFG represents the parts where control flow is determined. The data nodes are also incorporated in the CDFG. After the CDFG is constructed, the scheduler procedure prepares the schedule for the specification. This schedule takes in to account the mobility of the nodes of data flow graph, as well as the resources available.

The schedule is represented in the form of a list of states. Each state, in turn, is a list of binding structures that represent the operations from the DFG that are to be executed in that particular state. The binding structure contains information about the functional unit to which the operation is bound as well as the number of input and output buses required for the operation. It also defines whether a variable is being accessed from the memory or the register file.

The scheduler takes the availability of memory and register ports into consideration when preparing the schedule. Multi-cycling is used for operations with long delays. The scheduler is able to perform chaining of two or more operations with short delays. Many of the functional units considered are multi-functional, so the scheduler tries to choose the best option for binding to an operation.

4.2 Analysis and Refinement

The goal of analysis, as discussed previously, is to evaluate the performance of a given architectural specification in terms of power dissipation. This incorporates several steps.

4.2.1 Probability Analysis and Access Frequency

In the first step, the nodes of the CDFG as well as each DFG are assigned relative probabilities. For example, if a particular branch is executed 80 percent of the time whereas the

other branch is executed only 20 percent of the time, the relative probabilities of the two branches will be 0.8 and 0.2 respectively. For the purpose of our implementation, we follow a straightforward approach that assumes that all branches have equal branching probability and that all loops have finite and statically determined indices. Based on these assumptions, the probability of each CDFG node and corresponding DFG nodes is computed. This approach is fast and efficient.

An alternative approach supported by our current implementation is for the user to supply the probabilities (or typical execution frequencies) for each node of the CDFG corresponding to the behavioral specification. In that case, the user specified values are used for probability analysis. The user is prompted to choose an interactive probability insertion at the start; if a positive response is given, the program seeks probability for each CDFG node from the user, as the CDFG graph is traversed. The user can then provide appropriate values for each node.

Access Frequency: After the probability of execution is available, the next step is to traverse the behavior data structure (composed of CDFGs) in order to find out the probable number of accesses for the different functional units. This is done in the following manner: for each leaf behavior, the corresponding list of CDFG nodes is obtained. For each DFG structure on the list, the probability of execution is known as also the functional unit assigned to the operation. After traversing all the CDFG node structures, it is possible to compute the probable number of times that a given functional unit will be accessed. In other words, the access frequency for each unit is known. This information is stored in the behavior data struture.

4.2.2 Power Computation from Capacitance Models

Given the frequency of usage of the functional units, the next step makes use of the information in the allocation file (Subsection 4.1) that pertains to the average switched capacitance (per access) of each functional unit. This information is presently based on a series of simulations conducted at the transistor level for each functional unit [Agr96]. It may be noted that it is very easy to move from one library to another by simply changing the information in the allocation file. The implementation is independent of the library used.

By combining the frequency of usage and the capacitance information from the library, the power dissipated by a particular unit is computed. When the power dissipated for the buses, register and memory accesses, besides the functional units, is added up, this represents the estimated power requirement for the datapath components. This may then be used as a parameter for evaluation of the design.

4.2.3 Allocation Refinement

In the current version of implementation, the allocation refinement step is concerned only with trade-off changes. We attempt to obtain lower power dissipation at the cost of perfor-

mance as well as area. The replacement of functional units is based on their power dissipation. We will consider the implementation of optimization changes at a later stage.

Sorting of units: This step uses the information obtained in the previous step of analysis. The resources are arranged in a sorted list based on the maximum aggregate capacitance switched. This value depends upon the capacitance per access as well as the usage frequency. The element at the top of the list is selected for replacement.

Replacement of units: The resource library (which is different from the allocation file) is then scanned to get the same type of unit with lower capacitance. If replacement is available, the allocation file is modified to incorporate it and remove the original unit. This process continues till the resource list is exhausted or else user interrupts. At the end of refinement, a new schedule is prepared for the modified allocation and evaluated again.

4.3 Limitations of Delay and Area Estimation

As mentioned in Section 3.1, we assume that only the functional unit delay determines the clock cycle. This is not strictly true in practice, as register accesses, bus drivers and the interconnect wiring are also associated with finite delays. So while assigning the voltages and scaling the delay, we must keep in mind, that there are other components of the datapath, besides the functional units, that determine the clock cycle. For example, for an addition, two registers might need to be accessed, then buses are driven to supply data to the adder and the result bus carries the result data and loads it into another register, all within one clock cycle. Hence, when the voltage is reduced, delay for all these components will also increase, and the resulting minimum clock cycle value will increase, too. At this point, we are not considering these effects.

Another factor: generally, when the voltage is different, a different version of the component is used, rather than the same component. These different versions may have slightly different area as well as switched capacitance other than the difference in delay. However, we assume at this time, that the same component is used, so the switched capacitance and area remain the same.

A simple way to get around this would be to have different versions in the resource library for the same functional unit, corresponding to different supply voltages (these can be specified as a range, for more flexibility). Then, it would be possible to have more accurate values for area, switched capacitance as well as delay for a functional unit operation for a specific range of supply voltage.

5 Experimental Results

In this section, we present results obtained for several benchmarks for both steps of the design process, namely architecture selection and power estimation, using the implementation described in the previous section. For architecture selection paradigm, we experimented with common examples such as the

HAL differential equation solver; as well as the Kalman, Jacobian, elliptic wave and FIR filters. For the power estimation step, we carried out experiments on different versions of discrete cosine transform, which is commonly used in image compression algorithms. In the following sections, we discuss the examples and provide brief result analysis for each of them.

5.1 Experiments in Architecture Selection

The examples in this section relate to the architecture selection part of the design process. These were evaluated as a means of demonstrating the efficacy of the proposed design methodology. The component library is described in sub-section 4.1.1 and tabulated in Table 15 of Appendix C.

The general feature of architecture selection is that it is iterative in nature. Thus, the benchmarks are repeatedly scheduled for varying allocations (of functional units) and then analysed for the three parameters of performance, power and area. The probability approach used here is the simple default version described in sub-section 4.2.1, in which every branch at a forking point is assigned equal probability. After each analysis, refinement changes are carried out based on the direction of optimization required by the designer. This series iterates several times for fuller exploration of available architecture options. The parameters of clock period, execution time, and power dissipation are related in a complex fashion. Designer experience and intuition are useful for selecting the allocation as well as clock period.

The current evaluation of these benchmarks is limited to functional unit analysis. Though the tool implementation provides data about estimated number of memory, register and bus accesses, we consider only the functional units for determining power, execution time and area. The different design implementations may be suitable under different constraints. We shall remark upon the characteristics of each version and explain its significance and trade-offs involved. We do not attempt to make any decision, but leave it to the designer to select the implementation that best serves applicable constraints.

5.1.1 HAL Example

The first example studied is the HAL second-order differential equation solver. The specification as well as the DFG are provided in Appendix B. The steps to be followed are similar to the discussion above.

The results obtained for different iterations of the process of architecture selection are given in Table 1. Primarily, there were four different allocation choices, the difference being in multiplicity of units and the power consumption. Alloc A represents four functional units of type multiplier, adder, subtractor and comparator of the high speed version. Alloc B is same as Alloc A except for an additional high speed, high power multiplier. Alloc C represents the same units as Alloc A, but they are of the low power version. Alloc D represents the allocation of Alloc C with an extra low power multiplier.

Result Analysis: shows that each alloca-

Table 1: Evaluation of HAL Example

Alloc Type	Clk (ns)	Execution Time (ns)	Power (mW)	Alloc Area
A	50	1400	2.57	1150
B	50	1000	3.60	1750
B	60	900	5.26	1750
C	50	1450	1.84	800
D	50	1050	2.54	1250
D	70	1050	3.42	1250

Table 2: Elliptic Wave Filter

Alloc Type	Clk (ns)	Execution Time (ns)	Power (mW)	Alloc Area
A	50	1900	10.23	900
A	55	1760	11.05	900
B	50	1600	12.15	1500
B	55	1540	12.63	1500
C	50	1900	7.94	700
C	65	2080	7.25	700
D	50	1100	13.71	1150
D	65	1820	8.29	1150

tion has inherent advantages as well as drawbacks. Thus, Alloc A has moderately slow performance, but less power dissipation. On the other hand, if we compare it with Alloc C, we can get the same performance at the same clock period, at less power dissipation and lesser area. Alloc B has a very good performance, but the power cost is a bit high, especially at the clock period of 60 ns, when the number of states however is very less. It also requires the maximum area. Alloc C has minimum power but slow performance as compared to Alloc B or D, and its area cost is minimum. Alloc D has a reasonable power dissipation, good performance, and moderate area cost. So it seems to be a good choice for implementation.

5.1.2 Elliptic Wave Filter

For this example, the iterations resulted in four distinct allocations. Alloc A has two adders and a multiplier from the high speed library. Alloc B has an additional multipler unit. Alloc C is the low power counterpart of Alloc A, whereas Alloc D is the low power version of Alloc B.

Table 3: FIR Filter

Alloc Type	Clk (ns)	Execution Time (ns)	Power (mW)	Alloc Area
A	50	1850	9.74	750
A	55	1595	11.30	750
B	50	1150	15.67	1500
B	55	1045	17.24	1500
C	50	1850	7.39	570
C	65	1885	7.25	570
D	50	1150	11.88	1150
D	65	1235	11.07	1150

Result Analysis: The results are tabulated in Table 2. Alloc A has relatively low area, but high power dissipation, with moderate execution times. Alloc B reflects lower execution time, but at the expense of much more area and a little extra power than Alloc A. Alloc C has the minimum area as well as power dissipation, but a larger execution time. On the other hand, Alloc D provides a choice of higher execution time and low power versus faster execution and high power, just on account of changing the clock period. Its area requirement is also moderate.

5.1.3 FIR Filter

In this case, there were eight different allocation choices depending upon multiplicity of units and low power versions. However, for purposes of the table, these were pruned into four main allocations, following the same pattern as earlier. Alloc A and B differ in the multiplicity of units (B has two adders, two multipliers, whereas A has one of each). Alloc C and D also have the same difference, only the units are now from the low power library.

Result Analysis: Alloc A exhibits a significant change in execution time as well as power consumption when the clock frequency is altered. It does have very little area, and Alloc B though almost double in area, is much faster, with correspondingly high power dissipation. Alloc C provides the same performance as A, with lower power and area requirements. Similarly, Alloc D has the high performance of Alloc B, but for much lower power dissipation and significantly less area.

5.1.4 Kalman Filter

For this case, again there were four major allocation choices, with three different clock periods. Alloc A had only one multiplier, whereas Alloc B has two. Alloc C and D involve units from the low power library.

Result Analysis: For this example, Alloc C stands out in having low power consumption, fast execution time as well as the minimum area. It is not much behind Alloc B in

Table 4: Kalman Filter

Alloc Type	Clk (ns)	Execution Time (ns)	Power (mW)	Alloc Area
A	50	2600	54.4	900
A	55	2585	54.8	900
B	50	2450	57.8	1500
B	55	2475	57.2	1500
C	50	2650	41.1	700
C	65	3055	35.6	700
D	50	2500	43.6	1150
D	65	2925	37.2	1150

Table 5: Jacobian Filter

Alloc Type	Clk (ns)	Execution Time (ns)	Power (mW)	Alloc Area
A	50	17500	4.50	6500
A	55	16610	4.75	6500
B	50	13200	6.58	7200
B	55	14135	6.15	7200
C	50	17900	3.33	5000
C	65	19435	3.07	5000
D	50	13550	4.82	5400
D	65	16445	4.00	5400

performance, which is the fastest among all four. Only Alloc D has lower power dissipation, that also for a slow execution time. It is easy to point towards C as the optimum allocation choice.

5.1.5 Jacobian Filter

As with the other examples, we select four different allocations out of many possibilities. This particular example is interesting for its size (several hundred states) as well as the variety of functional units used. From the analysis, the access frequency of each unit was used as a pointer to determine if a particular unit should be replicated. From the same information, it was discovered that multiplications were very frequent, hence the pipelined multipliers corresponding to the two power versions were used in the allocation. As the results illustrate, this had a significant effect.

All the four allocations had identical number of units, except that Alloc A and C had non-pipelined multipliers (two each). Alloc C units were from the low power library. Alloc

B and D had pipelined multipliers (two each) from the standard and low power libraries respectively.

Result Analysis: Several conclusions are obvious. Alloc D has less area than Alloc A, but faster execution times at comparable or lesser power dissipation. Alloc B, has the fastest performance but high power and area requirements. Alloc C has the minimum power dissipation as well as area, but the execution times are relatively larger. On the whole, Alloc D might be a good choice, subject to the constraints of particular applications.

5.2 Experiments in Power Estimation

The earlier section dealt with experiments relating to architecture selection, and in this section we shall review some examples relating to the power estimation process. The reader is referred to Section 3.2 for the detailed discussion relating to this process. The important features of this section include:

- Accurate profiling information is taken into account for all the examples. Input data patterns are assumed to be random.
- The activity calculation takes into account the memory accesses, register accesses, bus loads, multiplexer usage as well as functional unit usage. Thus, as far as possible, all datapath elements are included in the estimation.
- We assume that flexibility in the choice of supply voltage is available to the designer. Thus we evaluate the design power dissipation for at least four different voltages, which in turn illustrates the important fact that lowering the supply voltage enables one to have better performance at low power.

Capacitance information is based on simulations done in [Agr96]. For this analysis, wiring or controller models are not used. We also assume that memory access time is one clock cycle, and that we have two read/write ports for memory. The delay scaling for voltage is done based on [CP96] and as given in Table 14.

As mentioned in the introduction to this section, the examples are different versions of the discrete cosine transform (DCT). This incorporates two consecutive matrix multiplications, a transformation that is very common for image processing algorithms. As such, DCT represents a big part of the applications in embedded systems that relate to image processing. We evaluate the primary sequential version first. Then we consider implementations that have optimizations for chaining of operations and allowing multicycling. Finally, we evaluate the loop-pipelined version, both with and without a pipelined multiplier. We analyse the results for each case, and compare them to results for other versions.

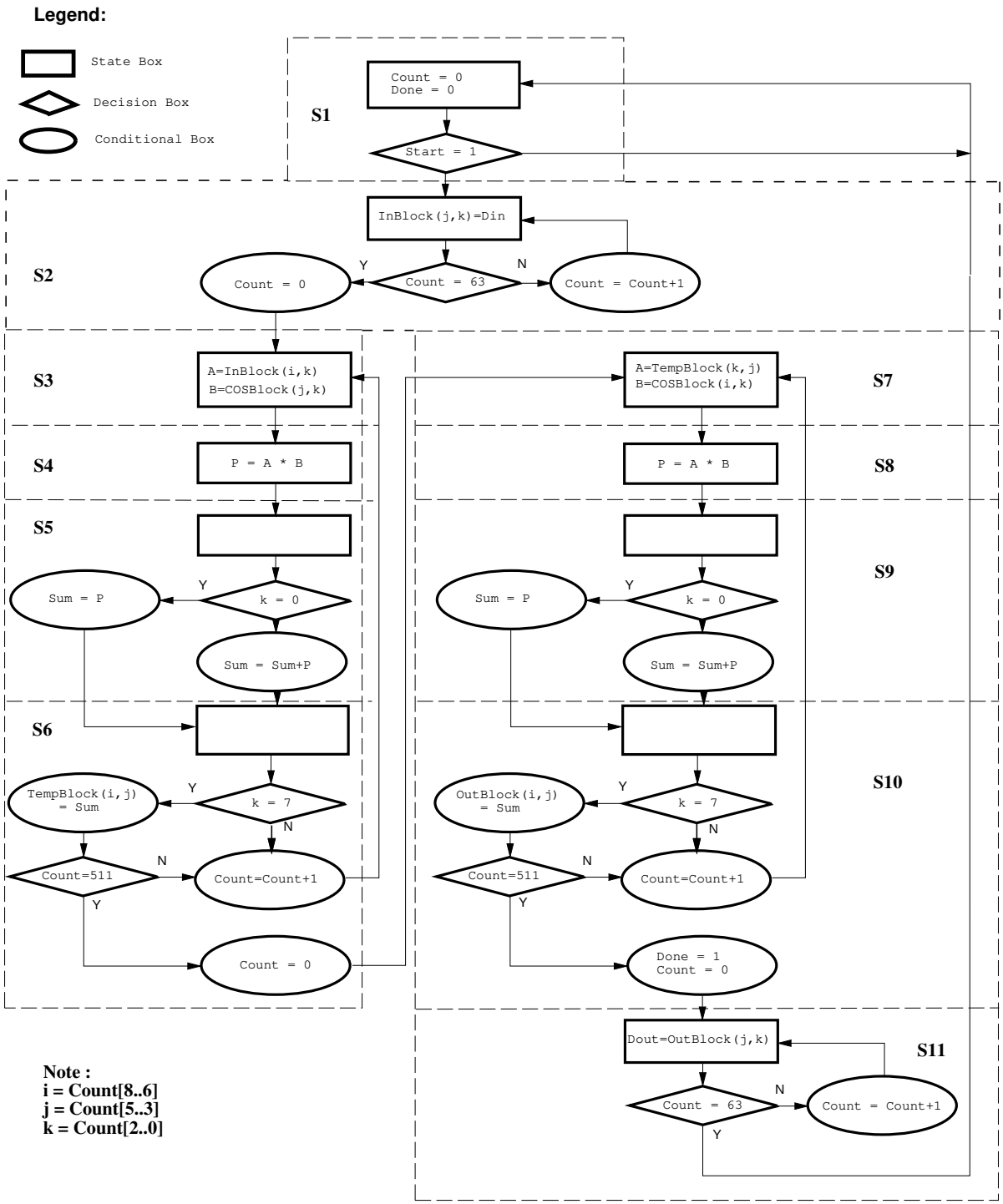

Figure 9: ASM Chart for Sequential Version of DCT

5.2.1 DCT: Sequential Version

The basic sequential version of DCT has two matrix multiplications that are performed in the steps shown in Figure 9. There are two input and output states, that relate to reading data from input memory and storing data into the output memory. There are eleven states in all, and states S2 and S11 correspond to memory input and output. They iterate sixty-four times, when the input is an eight by eight matrix. Similarly, the states from S3 to S6 and states S7 through S10, iterate 512 times to complete the multiplications. The results of evaluation are summarised in Table 6.

Table 6: DCT: Sequential Version

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	50	210	35.1
3.3	90	380	8.5
2.4	140	590	2.9
1.5	360	1520	0.4

Result Analysis: The results show that at 5V, the DCT sequential version takes around two hundred microseconds to complete execution, and dissipates at least 35 mW of power. The lower supply voltages result in lower dissipation of power, but execution time is also slowed down. Thus, if a 2.4V version dissipates only one-tenth of the 5V version, its execution time is almost thrice that of the latter.

5.2.2 DCT: Optimized Versions (no pipelining)

Generally, performance and image quality constraints would determine that the DCT be executed much faster. There are several ways in which one might try to optimize the implementation for better performance. It is relevant that the loops in DCT do not involve any data dependencies from one iteration to another. Hence the first optimization to be tried is loop unrolling. Also, we notice that some functional units like the adder have large slack, so we can try to combine more than one operation in a clock cycle. This is called chaining. Another approach that might be used is of multicycling, where a large delay operation (multiplication in this case) may be completed in two or more clock cycles, and the clock cycle itself can be reduced. All three of these optimizations may also be used in a combined form to get optimum implementations.

In the following sub-sections, we evaluate the optimizations and their effect upon performance versus power. At this point, we do not concentrate on area. The corresponding ASM charts for matrix multiplications as performed in the optimized versions are provided in Appendix D.

5.2.2.1 Loop Unrolling Since the DCT has two loops that are free from data dependencies, it is a good candidate for loop unrolling. Basically, instead of taking one data set at a time and multiplying it, we consider two of the data sets simultaneously. This reduces the number of iterations of the loop

by half. The evaluation results for the Loop Unrolled version of the DCT are provided in Table 7.

Table 7: DCT: Loop Unrolling

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	50	135	55.0
3.3	90	240	13.4
2.4	140	375	4.5
1.5	360	970	0.7

Result Analysis: On comparison with the sequential version, we find that the loop-unrolled version takes only about sixty percent of the execution time (at the same voltage levels), but the power dissipation also goes up correspondingly by about sixty percent. An interesting point is that if the supply voltage is decreased for this version to 3.3V, we can achieve lower power dissipation (almost one-third) at practically the same performance level as the sequential version at 5V.

5.2.2.2 Loop Unrolling and Chaining Next, we consider the loop unrolled version with the added optimization of chaining. This might be useful since in the unrolled loop form, there are two consecutive states that involve addition. Addition being much faster than multiplication (which determines the clock cycle), there is sufficient slack for both these operations to allow them to be combined in one state. The results of the evaluation are tabulated in Table 8.

Table 8: DCT: Loop Unrolling and Chaining

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	50	110	66.2
3.3	90	196	16.0
2.4	140	305	5.4
1.5	360	785	0.8

Result Analysis: Once again, on comparing the results for execution time for this version against the sequential version, we find a significant improvement in performance. In fact, this optimization has reduced the execution time further from the unrolled loop version. On the other hand, power dissipation has also increased, which would be logical. The highlight of this evaluation is that at a lower voltage of 3.3V, this version is faster than unrolled loop (3.3V), as well as sequential version (5V), with power dissipation only slightly higher than unrolled loop (3.3V) and much lower than sequential case at 5V.

5.2.2.3 Loop Unrolling and Multicycling Since we had encouraging results from applying the optimization of chaining, in addition to loop unrolling, we now look at another common optimization, i.e., multicycling. The rationale behind this is that the multiplier constrains a large value for the clock cycle, even though most of the other operations are much faster. So there might be some benefit from allowing two (or more) states for the multiplication operation and at the same time, reduce the clock cycle to minimize the slack.

Table 9: DCT: Loop Unrolling and Multicycling

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	25	80	92.4
3.3	45	144	22.4
2.4	70	224	7.6
1.5	180	576	1.2

Result analysis: For this case, the performance increases more than with combined chaining and loop unrolling. As expected, the power dissipation also increases. From this example, it seems that multicycling is more beneficial than chaining (from a performance viewpoint). We noticed that at 3.3V, this version has three times less power but similar performance as the unrolled loop version (5V). Also, this version at 2.4V is five times better in terms of power dissipation than the sequential version (5V) while providing the same performance.

5.2.2.4 Loop Unrolling, Chaining and Multicycling Since we had improvements in performance for both of the earlier optimizations of chaining and multicycling when applied separately to the unrolled loop version, it might be worthwhile to check if we can apply both of the optimizations at the same time. The point about which we have to be careful is whether the reduced clock cycle that is allowed by having multicycled operations, will still be long enough to chain two or more operations to one state. On checking that this was indeed valid in this case, we evaluated the combined version and the results are listed in Table 10.

Table 10: DCT: Loop Unrolling, Chaining and Multicycling

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	25	67	107.1
3.3	45	120	25.9
2.4	70	188	8.8
1.5	180	484	1.3

Result Analysis: This version was surprising in that the combined effect of the two optimizations was to reduce the execution time to half its value for the unrolled loop case. Interestingly, this version was also three times faster than the sequential version. In order to decrease the power dissipation, we can reduce the supply voltage. At 3.3V, this version has better performance than the loop unrolled version at 5V, and half of the corresponding power dissipation.

5.2.3 DCT: Pipelined Versions

Other than the optimizations of loop unrolling, chaining and multicycling, pipelining is also important. Pipelining has a benefit of being more universally applicable than either of the other optimizations. For data intensive applications such as DCT or other applications in the domain of image processing, pipelining allows designers to achieve higher performance at the cost of increased control complexity and area. It might be remarked that power also increases correspondingly. So, one might conclude that pipelin-

ing is not really good for power conservation. However, when combined with lower supply voltage, the power factor can be neutralised at a small performance cost.

Both the datapath as well as the functional units can be pipelined. The first example corresponds to datapath pipelining, without using any pipelined functional unit. The second example is a combination of datapath and functional pipelining, since we use a pipelined multiplier unit. The State Action Table for a loop pipelined matrix multiplication is given at the end of Appendix D.

5.2.3.1 Loop Pipelining For this case, the loops for both the matrix multiplications are pipelined into four stages. These four stages correspond to the load, multiply, add and store actions of the multiplication. As a result of the pipelining, the total time for the multiplication is reduced almost by a factor of four. Since the original number of iterations of each loop is quite large (512), this results in large performance gain. The clock cycle is still constrained by the maximum stage delay, which is in case of the multiplication. The results (Table 11) fulfil the prediction of large performance improvement.

Table 11: DCT: Loop Pipelining

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	50	58	150.8
3.3	90	104	36.5
2.4	140	162	12.4
1.5	360	417	1.9

Result analysis: This version is better than all earlier versions as far as performance is concerned. It will definitely be a preferred implementation for time-critical applications. However, power is somewhat on the higher side, which is to be expected. Datapath pipelining also involves higher area cost, which is not being considered for the present. Yet, by lowering supply voltages, it is still possible to get good performance as well as lower power dissipation. Thus at 2.4V, the power dissipation is lower and execution time is only two-thirds of the execution time for the unrolled loop version operated at 3.3V.

5.2.3.2

Loop Pipelining with Pipelined Multiplier Given that the loop-pipelined design has incredible performance, we go one step further and introduce functional unit pipelining for the multiplier, which is by far the component with the longest delay. We use a two stage pipelined multiplier, which results in halving the clock cycle. We note that the delay of the pipelined multiplier is slightly more than half of the non-pipelined version, and the same is true of switched capacitance and area. Since we are effectively accomplishing the same amount of activity in half the time, we might expect almost a two-fold gain in performance and at the same time, the power dissipation might be doubled.

Result Analysis: The hypotheses suggested in the earlier paragraph are well supported by the results, given in Table 12. As compared to the previous pipelined version, this one has half the execution time with little more than

Table 12: DCT: Loop Pipelining with Pipelined Multiplier

Vdd	Clk (ns)	T (us)	Power(mW)
5.0	25	29	327.6
3.3	45	52	79.3
2.4	70	80	27.0
1.5	180	210	4.1

double the power dissipation at each voltage level. However, as compared to the most optimized version, it consumes three times more power, with execution time that is barely 40 percent of the corresponding value. It is really amazing to note that this version is ten times as fast as the original sequential version, at the same voltage. As expected, power consumption is also ten times higher.

5.2.4 DCT: Result Analysis

The results tabulated above are compared together in various graphs in Figure 10 and Figure 11.

In the first figure, there are three curves corresponding to the sequential version of DCT, unrolled loop version and an optimized version with loop unrolling, multicycling and chaining. The points represent decreasing supply voltages from left to right from 5.0 V, 3.3V, 2.4 V to 1.5V.

In the second plot, the curves represent the optimized version and the loop pipelined versions (without and with the pipelined multiplier).

There are several significant facts that may be deduced from the results. Some of them

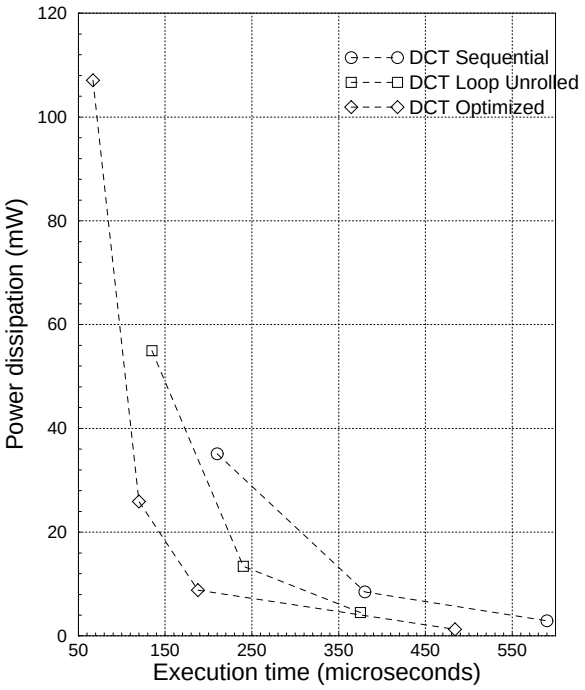

Figure 10: Comparison of Sequential, Loop Unrolled and Optimized DCT Versions

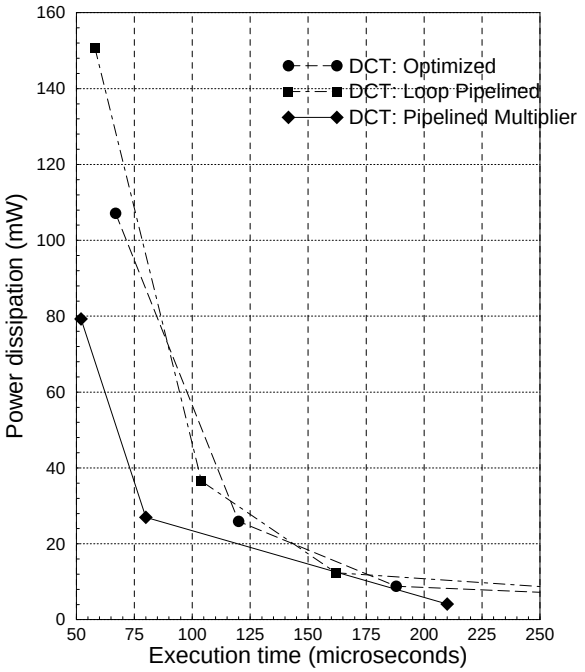

Figure 11: Comparison of Optimized and Loop Pipelined Versions of DCT

are:

- Loop pipelining is very successful in helping achieve high performance. However, other optimizations such as loop unrolling, multicycling and chaining can also significantly enhance performance.
- Detailed power/profiling estimation is necessary to able to compare designs that may be otherwise quite close in terms of performance and power dissipation. Thus, we are able to suggest from our results that loop-pipelined version is only slightly better in performance than the fully optimized version (with loop unrolling, chaining and multicycling). This might be difficult to conclude with only an approximate analysis.
- Power dissipation is directly related to performance, as long as supply voltage is constant. So if we gain something in terms of performance, we lose the same in terms of power.
- Supply voltage has been demonstrated to have considerable effect on power dissipation. In fact, it is an important method to lower power dissipation, while not paying the entire cost in terms of performance.
- Given that assignment of supply voltage greatly affects the power dissipation, it is necessary to have a design environment that allows different modules to operate at different supply voltages.

6 Conclusions

In the preceding sections, a novel two step iterative methodology has been proposed to enable behavioral exploration for low power designs. The examples corresponding to each step, illustrate the benefit of this approach for designers that incorporates two-level power estimation. From the results in Section 5, the following features of the methodology are demonstrated:

Interactive environment: Designer input and control of design flow is possible in each iteration. This is useful since complex trade-offs are involved in power optimization.

Quick analysis: Fast and efficient design iteration enables the user to explore larger design space.

Trade-off Visualization: The analysis in both the steps provides extensive information for the designer to have clear perception of design space relationships and trade-offs for informed decision-making.

Dependence on Resource Models: The accuracy of results depends on the proper characterization of underlying component models and libraries. As characterization of libraries improves, we will have more accurate results.

Flexible nature: We allow the designer to control the design flow direction as well as the design iterations. This helps in shortening design time, as well as better designs on account of added input of designer intuition and experience.

We conclude that our implementation meets the initial targets (refer Section 1.3) of allowing fast design space exploration, interactive nature, easily modifiable allocation and of providing important design information to the user.

Future Work

As far as architecture selection process is concerned, there are several enhancements that need to be considered. The analysis part has to be extended to incorporate loop bounds, and pipelined datapaths. Designs having a fixed rate of data input and output will also be focussed upon. For allocation refinement, we need to consider functional units with available slack also, as these units can yield potential benefits for power dissipation without performance degradation.

For the power estimation section, the extensions possible are realistic controller and wiring (interconnect) models. Also, we need to link up with a physical layout and simulation system, in order to compare the relative accuracy of analysis results with actual figures. We hope to compile a more comprehensive resource library with multiple levels of components, based on performance and power dissipation. This should help in providing wider range of possible implementations and enhance the granularity of variations in different parameters of the design.

We also plan to expand our suite of benchmarks, so as to have more complex examples of typical embedded system applications. With suitable enhancements as suggested above, we plan to develop the present implementation into a robust framework for a high level synthesis system targeting optimization for power.

Acknowledgement

This work was supported by Toshiba research grant TC 21534. Figure 9 (ASM chart for sequential version of DCT) and other flowcharts for different DCT examples are taken from course materials for ICS 155, Dept. of Information and Computer Science at University of California, Irvine. Thanks are due to Gaurav Aggarwal for helping with figures; we also appreciate the meaningful insights and useful suggestions given by Jianwen Zhu and Poonam Agrawal.

References

[Agr96] Poonam Agrawal. "PRESTO: A Gate-level Power Analyser,". Master's thesis, University of California, Irvine, 1996.

[CP96] Jui-Ming Chang and Massoud Pedram. "Energy Minimization Using Multiple Supply Voltages,". In *International Symposium on Low Power Electronics and Design*, 1996.

[Gaj95] Daniel D. Gajski. *Principles of Digital Design.* Prentice Hall, New Jersey, 1995.

[GVNG94] Daniel Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and Design of Embedded*

Systems. Prentice Hall, New Jersey, 1994.

[GWDL94] Daniel Gajski, Allen Wu, Nikil Dutt, and Steve Lin. *High Level Synthesis*. Kluwer Academic Publishers, Massachussetts, 1994.

[KKRV96] Nand Kumar, Srinivas Katkoori, Leo Rader, and Ranga Vemuri. "Profile-driven Behavioral Synthesis for Low Power VLSI Systems,". In *IEEE Design and Test of Computers*, 1996.

[Lan96] Paul Landman. "High Level Power Estimation,". In *International Symposium on Low Power Electronics and Design*, 1996.

[LMR95] Paul Landman, Renu Mehra, and Jan Rabaey. "An Integrated CAD Environment for Low Power Design,". In *IEEE Design and Test of Computers*, 1995.

[LR93] Paul Landman and Jan Rabaey. "Power Estimation for High Level Synthesis,". In *European Design Automation Conference and Euro-ASIC*, 1993.

[LR95] Paul Landman and Jan Rabaey. "Architectural Power Analysis: The Dual Bit Type Method,". *IEEE Transactions on VLSI Systems*, pages 173–187, Jun 1995.

[MR94] Renu Mehra and Jan Rabaey. "Behavioral Level Power Estimation and Exploration,". In *First International Workshop on Low Power Design*, 1994.

[Naj87] Farid N. Najm. "A Survey of Power Estimation Techniques in VLSI Circuits,". *IEEE Transactions on VLSI*, pages 446–455, Dec 1987.

[RJ94] Anand Raghunathan and Niraj K. Jha. "Behavioral Synthesis for Low Power,". In *International Conference on Computer Design*, 1994.

[SR96] Sergei Sokolov and Anand Raghunathan. "VHDL Behavioral Synthesis tool for Low Power based on FRITS,". Technical Report SBIR Phase-I, ASC, Princeton University, 1996.

[ZAG97] Jian-Wen Zhu, Poonam Agrawal, and Daniel Gajski. "RT Level Power Analysis,". In *Asia-Pacific Design Automation Conference*, 1997.

A SRA Algorithm Description

A flowchart illustrating the computation for the SRA algorithm is provided. The functional units used are absolute unit, max, min, adder and subtractor. The shifts are accomplished through appropriate connections. The two integers are read from two input ports and the result is written to an output port.

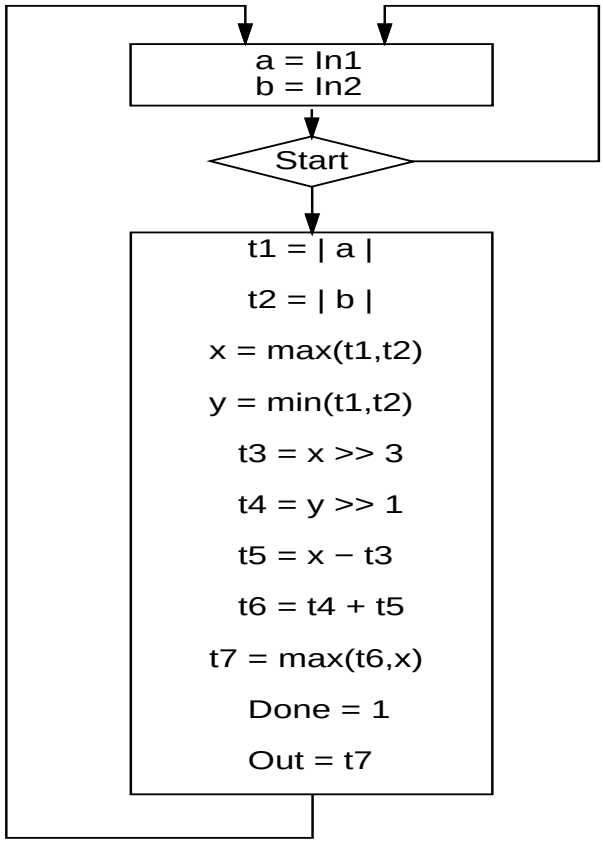

Figure 12: Square Root Algorithm Flowchart

B HAL Specification and DFG

The DFG as well as the textual description of the HAL example is given in Figure 13. This is a second order differential equation solver.

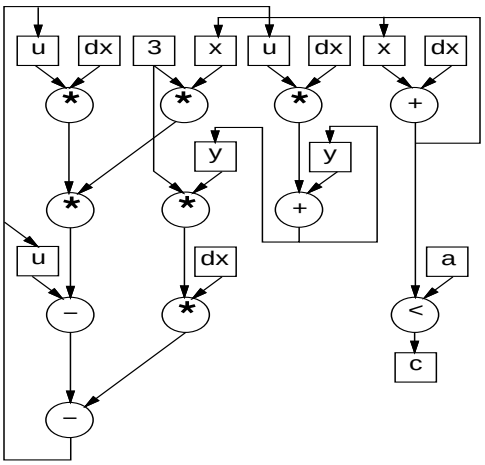

Figure 13: HAL: DFG and Textual Description

C Component Library

Component Library for Examples: The components used in the examples given in Section 3.1.1 and Section 3.1.2 are characterised in Table 13. The values for the average switched capacitance per access were generated using PRESTO [Agr96]. The components were generated using 0.6 micron, 3.3V VTI standard cell CMOS library. These components are synthesized using only two input NAND and NOR gates and inverters.

Delay Voltage Curve: Table 14 provides a scaling of functional unit delay with change in supply voltage. This has been based on [CP96].

Component Library for Implementation: The different components used by the implementation are listed in Table 15. The important characteristics of the components include delay, average switched capacitance per access, area and bitwidth. There are also fields for specifying the number of control lines, the multiplicity and the pipelined nature of the units. The allocation file is based upon the units present in the component library.

Table 13: Component Library for Examples

Unit	Cap./access (pf)	Delay (ns)	Area (trans)	Bitwidths
mod (ripple carry)	3.82	16.4	188	4
max (" ")	3.82	17.8	196	4
min (" ")	3.82	17.8	196	4
adder (" ")	3.18	12.6	124	4
subtr (" ")	3.36	14.0	132	4
array mult (" ")	59.2	117.4	2656	8
booth mult (" ")	58.6	109.6	2608	8
mod (carry look-ahead)	4.07	11.4	326	4
max (" ")	4.07	12.8	340	4
min (" ")	4.07	12.8	340	4
adder (" ")	3.43	9.0	196	4
subtr (" ")	3.54	10.4	208	4
array mult (skew carry)	71.6	61.6	3213	8
booth mult (" ")	79.4	56.2	3562	8

Table 14: Delay vs. Supply Voltage

Vdd (volts)	Delay
5.0	1.0
3.3	1.7
2.4	2.8
1.5	7.2

Table 15: Component Library for Implementation

Unit	Delay (ns)	Ave. Cap./access (pf)	Area	Bitwidth
abs	56	15.9	144	16
abs	67	11.9	101	16
adder	49	13.8	144	16
adder	59	10.3	101	16
comparator	23	13.3	112	16
comparator	28	10.0	78	16
divide	163	401.8	608	16
divide	196	301.3	426	16
max	23	13.3	112	16
max	28	10.0	78	16
min	23	13.3	112	16
min	28	10.0	78	16
mod	163	401.8	608	16
mod	196	301.3	426	16
multiplier	163	401.8	608	16
multiplier	196	301.3	426	16
rem	163	401.8	608	16
rem	196	301.3	426	16
subtractor	56	15.9	288	16
subtractor	67	12.0	202	16

D ASM Charts for DCT

The ASM charts for different versions of unrolled loop matrix multiplication (as used in DCT) are given next, followed by a State Action Table for Loop pipelined matrix multiplication.

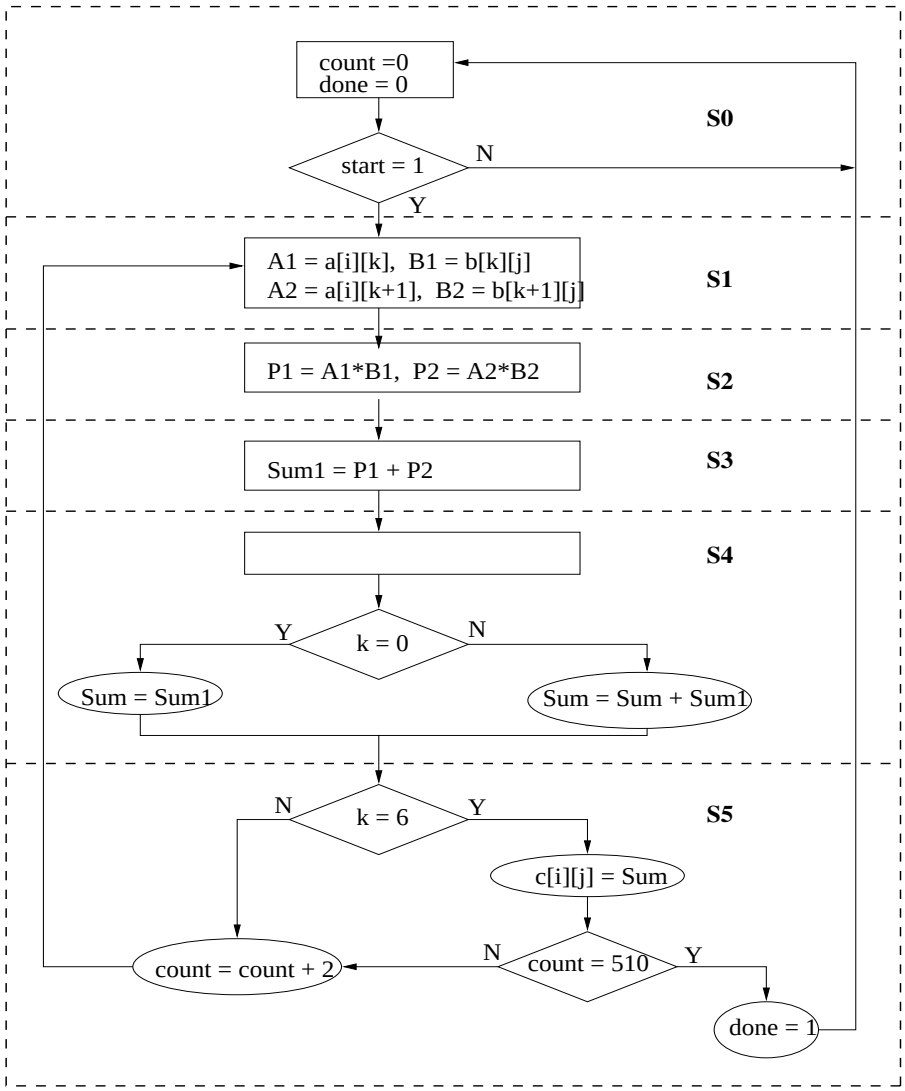

Figure 14: Matrix Multiplication with Loop Unrolling

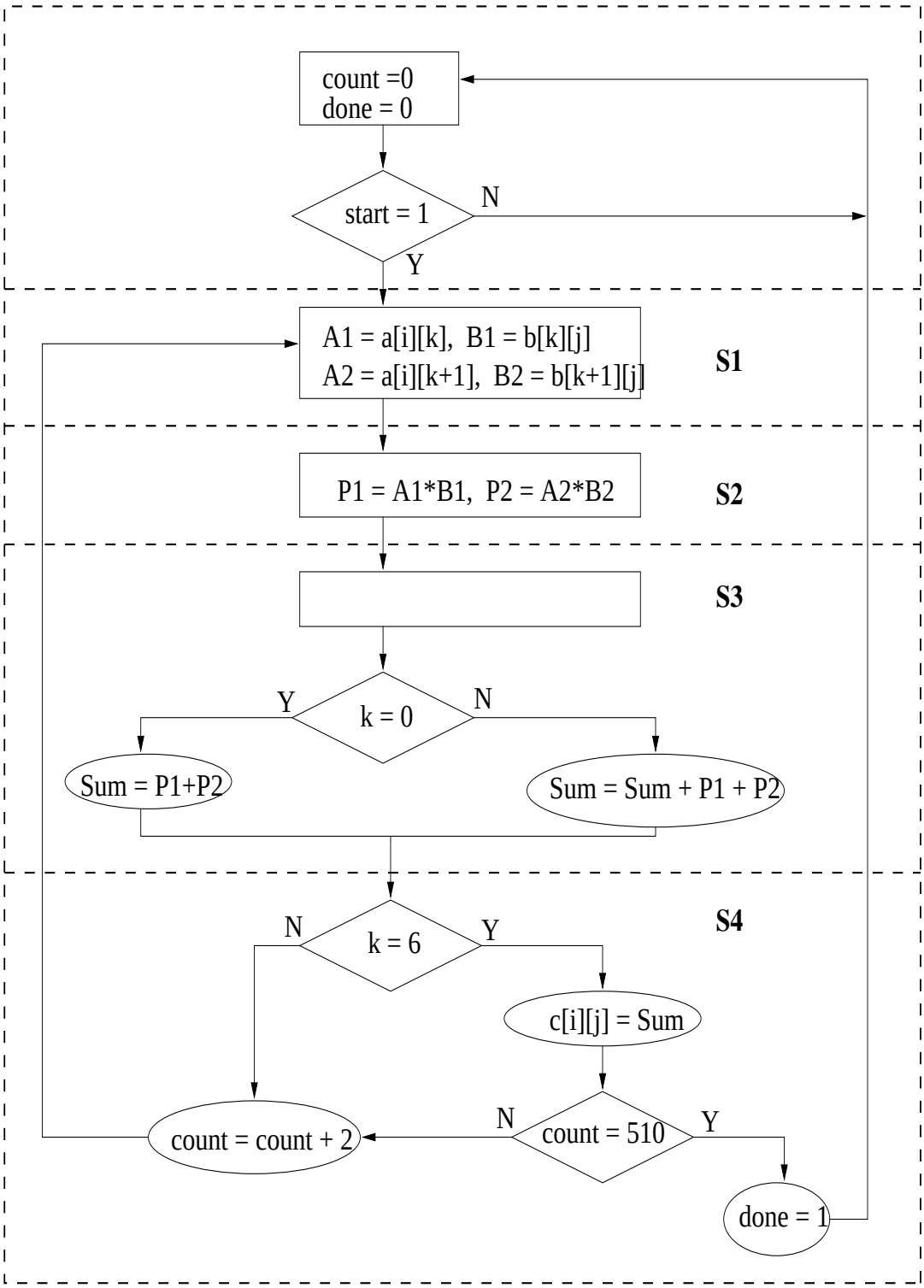

Figure 15: Matrix Multiplication with Loop Unrolling and Chaining

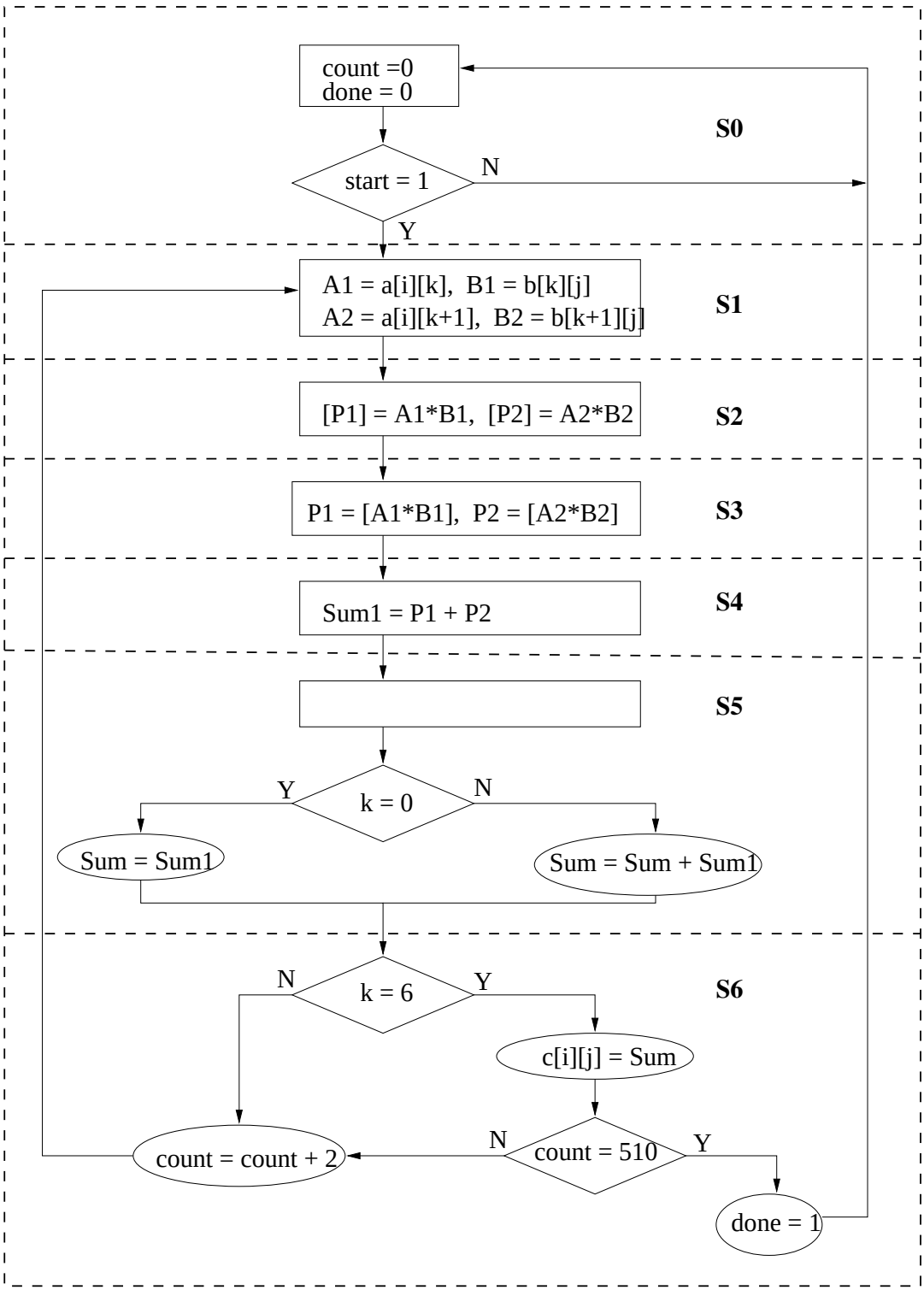

Figure 16: Matrix Multiplication with Multicycling and Loop Unrolling

STATE ACTION TABLE
for Pipelined Matrix Multiplication

PRESENT STATE	NEXT STATE CONDITION, STATE			CONTROL AND DATAPATH ACTIONS CONTROL, ACTIONS
S0 (000)	Start=0, Start=1,	S0 S1	000 001	count0=0 done=0
S1 (001)		S2	010	A=a[i0][k0] B=b[k0][j0] count1=count0 count0=count0+1
S2 (010)		S3	011	P=A*B count2=count1 A=a[i0][k0] B=b[k0][j0] count1=count0 count0=count0+1
S3 (011)		S4	100	k2=0, Sum=P k2≠0, Sum=Sum+P count3=count2 P=A*B count2=count1 A=a[i0][k0] B=b[k0][j0] count1=count0 count0=count0+1
S4 (100)	count0≠511, count0=511,	S4 S5	100 101	k3=7, c[i3][j3]=Sum k2=0, Sum=P k2≠0, Sum=Sum+P count3=count2 P=A*B count2=count1 A=a[i0][k0] B=b[k0][j0] count1=count0 count0≠511, count0=count0+1
S5 (101)		S6	110	k3=7, c[i3][j3]=Sum k2=0, Sum=P k2≠0, Sum=Sum+P count3=count2 P=A*B count2=count1
S6 (110)		S7	111	k3=7, c[i3][j3]=Sum k2=0, Sum=P k2≠0, Sum=Sum+P count3=count2
S7 (111)		S0	000	k3=7, c[i3][j3]=Sum done=1

D0 = (Start AND S0) OR S2 OR ((count0=511) AND S4) OR S7
D1 = S1 OR S2 OR S5 OR S6
D2 = S3 OR S4 OR S6 OR S7

Figure 17: Loop Pipelined Matrix Multiplication