High Level Profiling Based Low Power Synthesis Technique

Srinivas Katkoori, **Nand Kumar** and **Ranga Vemuri**

Address for Correspondence:
Dr. Ranga Vemuri, Director
Laboratory for Digital Design Environments
Department of Electrical and Computer Engineering
813 Rhodes Hall; Mail Location 30
University of Cincinnati
Cincinnati, Ohio 45221-0030
Phone: (513)-556-4784
Fax: (513)-556-7326
Email: ranga.vemuri@uc.edu

Appeared in International Conference on Computer Design (ICCD) 1995

This work is done at the University of Cincinnati and is supported in part by the Solid State Electronics Directorate of the Wright Laboratory of the US Air Force under contract number F33615-91-C-1811 and by the Advanced Research Projects Agency under order no. 7056 monitored by the Federal Bureau of Investigation under contract no. J-FBI-89-094.

High Level Profiling Based Low Power Synthesis Technique

Abstract

We present a profiling based technique for power estimation. This technique is implemented in the PDSS (Profile Driven Synthesis System) for the synthesis of low power designs. Initially, each module in the module library is characterized for the average switching capacitance per input vector. The input description is simulated using user-specified set of input vectors to collect the *profile data* for various operators and carriers. The *profile data*, in conjunction with the pre-characterized module library is used to estimate the total capacitance switched by each of the valid schedules produced by the PDSSmbox mbox scheduler. A valid schedule is one which satisfies other constaints such as area and delay. The schedule with the least switching capacitance estimate is further synthesized to the layout level. Results show an average deviation of 12% compared with the actual switching capacitance values at the layout level.

High Level Profiling Based Low Power Synthesis Technique

1 Introduction

As the complexity of VLSI Systems is rapidly increasing, power consumption is becoming a major design constraint. Since power estimation at the lower levels of design abstraction is very expensive, power estimation at the behavioral level is an attractive solution. In this paper, we present a high level technique for power estimation. PDSSmbox mbox (Profile Driven Synthesis System) employs this technique to synthesize low power designs.

High level synthesis is the process of generating a register level design from an algorithmic behavioral specification. A typical synthesis system is shown in (Figure 1). The inputs to the system are the behavioral specification, a module library and the user constraints. The behavioral specification can be written in a high level general purpose language like C or in a Hardware Description Language like VHDL. The module library consists of storage units like registers and memories, execution units like adders and multipliers, and interconnect units like multiplexors and buses. Typical user constraints are area, clock speed and power. Traditionally, the output of the synthesis system consists of two interacting components namely *data path* and *controller*. The data path is built from the modules of the module library and the controller is a finite state machine to be implemented either as a PLA or a microprogram. High level synthesis is followed by logic and layout synthesis resulting in fabricatable mask layouts targeted for various technologies.

A profile-based approach for low power behavioral synthesis is presented in this paper. Initially, for each module in the parameterized module library and for each size, average switching capacitance per input vector is determined as a function of the bit-width of the module. This constitutes the characterization of the module library. The synthesis flow of Profile Driven Synthesis System (PDSS), is explained briefly as follows: The input specification is simulated using user-specified set of input vectors to get the profile data for the various operations and carriers. The profile data and the switching capacitance characteristics from the module library, are used to get an estimate of the aggregate switching capacitance for each of the schedules that satisfy the area and clock speed constraints. The schedule with the least estimate is synthesized to the layout level. We assume that the target is CMOS technology.

We synthesized a variety of designs consisting of various behavioral level constructs like conditionals, loops

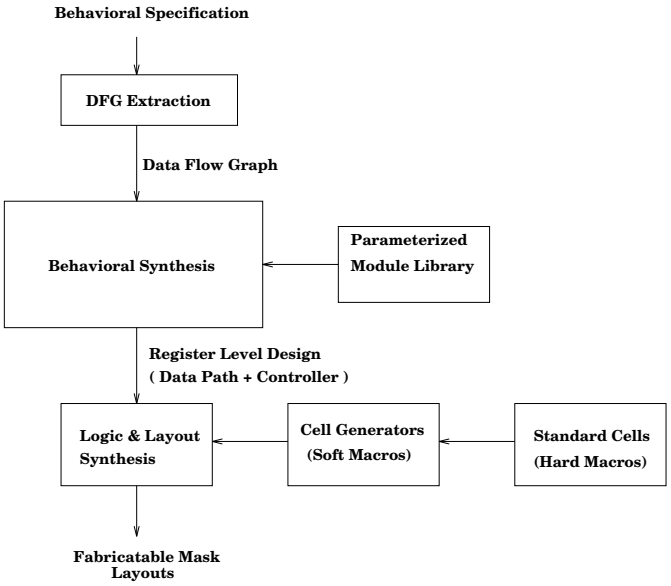

Figure 1: **A Typical High Level Synthesis System**

and subprograms etc. On an average, the estimates of the switching capacitance is within are 12% of the actual switching capacitance measured by simulating the switch level models from the layouts.

2 Previous Work

Power estimation techniques at the gate and lower levels of abstraction can be broadly classified [17] into (1) *simulation* based techniques; (2) *probabilistic* techniques; and (3) *statistical* techniques. Typically, in a *simulation* based technique [23, 25, 28] the average power is calculated by monitoring either the supply voltage or current waveforms or both. These are too slow to handle very large circuits. In a *probabilistic* technique [34, 18, 36], user-supplied input signal probabilities are propagated into the circuit. Various approaches based on different probabilistic measures such as transition density [16] are proposed. In a *statistical* technique [2, 3], the circuit is simulated with randomly generated input vectors until power

converges to the average power where convergence is tested by statistical mean estimation techniques. At the architectural level, Landman *et al* [8] presented a technique for the characterization of module library using signal statistics. At the behavioral level, Chandrakasan et al., [4, 5] present a high-level synthesis system, HYPER-LP, which minimizes power consumption in application specific data path intensive CMOS circuits using a variety of architectural and computational transformations. Powell et al. [7] presented a high-level power dissipation model for DSP algorithms. Renu et al. [6] applied a combination of analytical and stochastic techniques for power estimation.

3 Profile-Driven Synthesis System

The architecture of PDSS is shown in Figure 2. PDSS accepts specifications in a behavioral subset of VHDL and constraints in terms of clock-period and area. It generates a constraint-satisfying design with the least amount of estimated switching capacitance. The behavioral specification is first translated into a data flow graph (DFG) representation which captures both the data and control flow information among the operations and carriers in the specification. Figure 3 shows a toy specification and its DFG representation. The DFG contains *operation* nodes which represent arithmetic, relational and boolean operations and *control* nodes which represent conditional statements. Cycles in the DFG denote iterative statements in the specification. Each edge in the DFG denotes data flow or control flow. In this paper, we assume that procedure and function calls have been expanded in-line.

Characterization of Module Library for ISC: The module library contains parameterized register-level modules such as n-bit registers, n-bit adders and n-bit m-to-1 multiplexors. Modules are parameterized with respect to number of inputs where applicable and bit-width of each input, and characterized for performance attributes such as area, delay and average intrinsic switching capacitance (ISC). These attribute values are determined by actually generating layouts for several instances of the module with different parameter values. Area can be directly measured from the layout and delay can be determined through simulation or a by timing analysis. Determination of ISC which depends on input patterns is more involved and is described below.

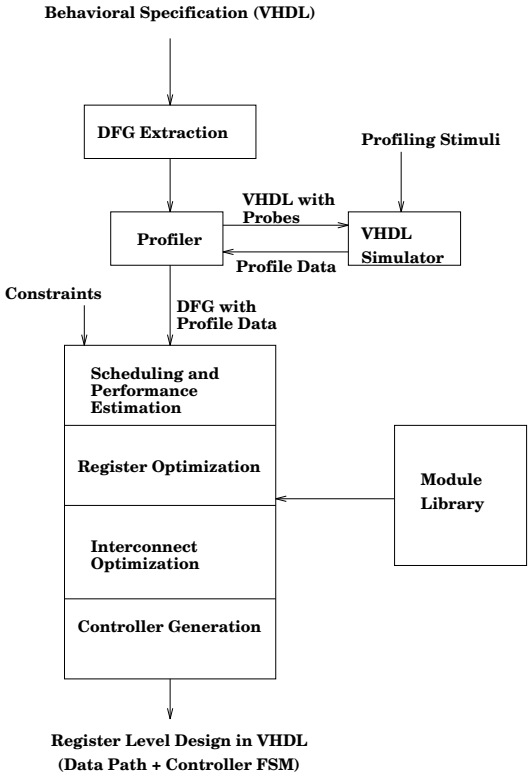

Figure 2: **Profile Driven Synthesis System (PDSS)**

```
ENTITY toy IS
  PORT(a,b : IN INTEGER;
       --%PRAGMA BIT_WIDTH 4
         c : OUT INTEGER);
       --%PRAGMA BIT_WIDTH 5
END toy;

ARCHITECTURE foo OF toy IS
BEGIN
 p : PROCESS(a,b)
   VARIABLE u,v : INTEGER;
       --%PRAGMA BIT_WIDTH 5
 BEGIN
   u := a + b;
   v := a - b;
   IF(a > b)
    THEN
     c <= u;
    ELSE
     c <= v
   END IF;
 END PROCESS;
END foo;
```

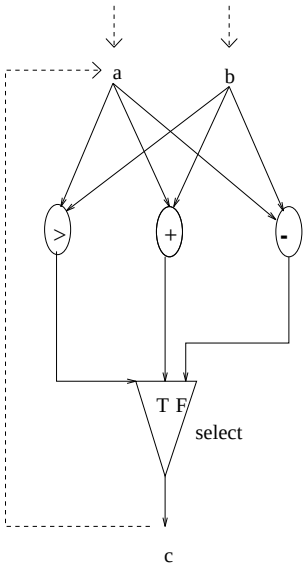

Figure 3: **A VHDL Behavioral specification and its DFG**

Intrinsic Switching Activity, ISC of a module instance is the average capacitance that is expected to switch when an input event (change of logic values on the input lines) takes place.

ISC of a module instance is determined by extracting a switch level model from its layout, simulating the switching level module using a very long stream of randomly generated input patterns and monitoring the capacitance switched per pattern. Simulation is done using a modified version of IRSIM []. Simulation and counting continues until convergence occurs as defined below.

Let C_k be the total capacitance switched after applying k random input patterns without reinitialization between successive patterns. $Z_k = \frac{C_k}{k}$ denotes the average switching capacitance per input pattern after applying k patterns. $\delta_k = \frac{|Z_k - Z_{k-1}|}{Z_{k-1}}$ denotes variation in the average capacitance between the $k-1$ th and k the patterns.

We continue to apply random input patterns until δ_k remains less than 0.001 over 1000 consecutive input pattern applications. At this point we say that the average switching capacitance estimation converged and accept the value of Z_k after the last input pattern is applied. This value is the ISC of that instance of the module. Similar procedure is used to determine the ISC of various instances of the module and results are expressed as a table.

Figure 4 shows the ISC plot with respect to the bit-width for three modules namely, adder, register and two-input multiplexor. Table 2 shows the ISC characteristics for some of the PDSS library modules. Linear interpolation or extrapolation will be used for bit-widths not included in Table 2.

PLA Characterization: PDSS assures that the controller will be implemented by a CMOS PLA Structure. A PLA is characterized by three parameters : (1) *input size* , $\mathcal{I}$; (2) *output size*, $\mathcal{O}$; and (3) the *number of states*, $\mathcal{S}$. Thus the ISC for any controller is a function of these parameters. It is observed that, for any random input vector, a PLA switches approximately the same amount of capacitance for each clock step. By varying three PLA parameters $(\mathcal{I}, \mathcal{O}, \mathcal{S})$, random PLAs are generated and are subjected to random input vectors. For each $(\mathcal{I}, \mathcal{O}, \mathcal{S})$ combination, the Intrinsic Switching Capacitance per clock step (as opposed to per input pattern in the case of module library) is determined. The simulation is continued until the ISC converges in a manner similar to the determination of ISC for the module library as discussed above. Thus a PLA characterization table is computed which is used later for the estimation of switching capacitance in a controller. A portion of this table is as shown in Table 1

Sl.No	$\mathcal{I}$	$\mathcal{O}$	$\mathcal{S}$	**ISC(pF)**
1.	3	24	14	16.04
2.	3	26	21	26.57
3.	4	7	12	8.70
4.	4	16	16	17.10
5.	4	18	8	11.01
5.	4	22	8	12.35
6.	5	10	14	12.51
7.	6	5	11	10.39
8.	6	6	9	7.70
9.	6	7	15	11.70
10.	6	12	16	14.03
11.	6	14	16	15.21
12.	6	30	23	37.41
13.	7	36	13	31.48
14.	10	21	48	48.75
15.	10	18	47	42.75

Table 1: **A portion of the PLA Characterization table**

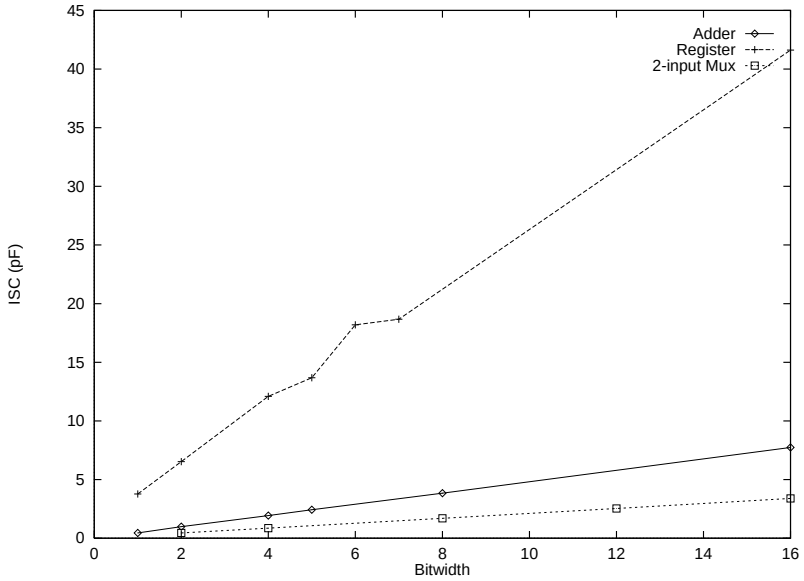

Figure 4: **ISC vs. Bit-Width for three Parameterized Modules**

Sl.	Module	ISC Table (BitWidth–ISC(pF))
1.	Adder	1–0.45, 2–0.98, 4–1.93, 5–2.43, 8–3.84, 16–7.74
2.	Subtractor	2–0.97, 4–2.50, 6–3.26, 8–5.64, 10–7.05, 16–12.16
3.	Comparator (>)	1–0.44, 2–0.88, 4–1.82, 5–2.00, 6–2.78, 8–3.99, 16–12.57
5.	Multiplier	2–2.27, 3–3.53, 4–7.99, 5–15.30, 8–60.48, 16–45.39
6.	Multiplexor	2-inputs: 2–0.45, 4–0.86, 8–1.70, 12–2.53, 16–3.39 4-inputs: 2–1.41, 4–2.68, 8–5.20, 12–7.95, 16–10.79 6-inputs: 2–2.46, 4–4.69, 8–9.46, 12–14.53, 16–19.53 8-inputs: 2–3.29, 4–6.23, 8–13.10, 12–19.89, 16–26.73
7.	Register	1–3.77, 2–6.53, 4–12.09, 5–13.68, 6–18.19, 7,18.67, 16–41.62
8.	Signal Register (Register + Glue Logic)	2–10.90, 3–12.41, 4–15.45, 5–15.99, 8–23.78, 16–39.35
9.	AND	2–0.17, 3–0.29, 4–0.36, 5–0.45, 6–0.55, 8–0.76, 10–0.97, 16–1.55
10.	OR	2–0.18, 3–0.27, 4–0.38, 5–0.48, 6–0.51, 8–0.71, 10–0.98, 16–1.48
11.	NOT	1–0.04, 2–0.08, 3–0.12, 4–0.16, 5–0.20, 8–0.33, 16–0.66
12.	NAND	1–0.06, 2–0.13, 3–0.19, 4–0.26, 5–0.32, 6–0.38, 7–0.44, 8–0.53, 16–1.06
13.	NOR	3–0.17,4–0.22,5–0.28,6–0.35,8–0.47
14.	XOR	2–0.31, 3–0.50, 4–0.68, 5–0.86, 6–0.98, 8–1.35, 10–1.69
15.	XNOR	2–0.31, 3–0.50, 4–0.64, 5–0.80, 6–0.97, 8–1.26, 10–1.61, 16–2.56

Table 2: **ISC Data for Some Parameterized Library Modules** (BitWidth ≥ 1)

Profiler: Profiler takes the DFG representation of the specification and generates equivalent VHDL program with probes (counters and similar monitoring variables) to gather various event activities. We need to profile the DFG rather than the original specification, since the DFG representation exposes all the operations and carriers (edges in DFG) that will be bound to hardware resources. The generated VHDL program is simulated using input vectors called the *profiling stimuli*. Profiling stimuli should represent typical usage of the design being synthesized. Since profiling stimuli will decide the event activity in the design and will guide the synthesis system in generating a low power design, the user should take extreme care in preparing this test data. We recommend the following methods for preparing profiling stimuli, based on the type of design being synthesized:

- *Arithmetic Dominated Specifications:* Profiling stimuli should include a relatively large number of input patterns representative of various regions of input space.
- *Control Dominated Specifications:* Profiling stimuli should be selected to execute typical control flow paths through the specification. Behavior level stimulus generators that guarantee execution of desired paths through the specification can be used for this purpose [10, 11].
- *Instruction Set Processors:* Profiling stimuli for these specifications are best generated using assemblers and compilers by translating typical high level language programs, such as editors, text processors, operating system kernels etc. into machine code.

The goal of profiling is to gather the following data:

- For each DFG node *op*, determine the number of times the node is executed for the given profiling stimuli. This number is called the event activity of the operation node and is denoted by E_{op}.
- For each edge e, determine the number of times the edge is traversed during execution. This number is called the *transaction activity* of the edge and is designated by T_e.
- For each edge e, determine the number of times the value on the edge has changed. This number is called the *event activity* of the edge and is denoted by E_e. Note that $E_e \leq T_e$.

Time	a	b
1	5	7
2	10	3
3	15	11
4	11	11

Figure 5: **Sample Profiling Stimuli for the Example in Figure 3**

Probes are inserted by the profiler to measure the above quantities, collectively called the *profile data* of the specification. Nodes and edges in the DFG are annotated with the profile data. Annotated DFG is then submitted for synthesis. Figure 5 shows sample profiling stimuli for our example and Figure 6 shows the DFG annotated with the profile data. For edges, the number in the parenthesis denotes event activity and the number outside the parenthesis denotes transaction activity.

Scheduling and Performance Estimation: Following profiling, there are four major phases in PDSS: scheduling, register optimization, interconnect optimization and control generation. During scheduling, abstract operations in the DFG are assigned to control steps and various operation nodes are bound to specific instances of modules selected from the module library. A schedule is acceptable provided the estimates of area and clock period satisfy the user-specified constraints. Scheduling itself is performed to obtain a minimal length schedule subject to a resource constraint, namely, only the modules available in a module bag should be used. [1] A module bag is a collection of instances of modules from the module library. A valid module bag satisfies the following properties: (1) For each operation in the DFG, the bag has at least one module instance which implements the operation; (2) The delay of each module in the bag is no more than the constraint on clock period. We assume non-pipelined designs; (3) The total area occupied by the module instances in the bag should not exceed the area constraint. Clearly, different valid module bags lead to different schedules and hence different tradeoff points in the design space. An efficient synthesis system generates a variety of valid module bags from the library, generates corresponding near-minimal

[1] A *bag* is a collection of items with duplicates allowed. A *set* is a collection items with no duplicates.

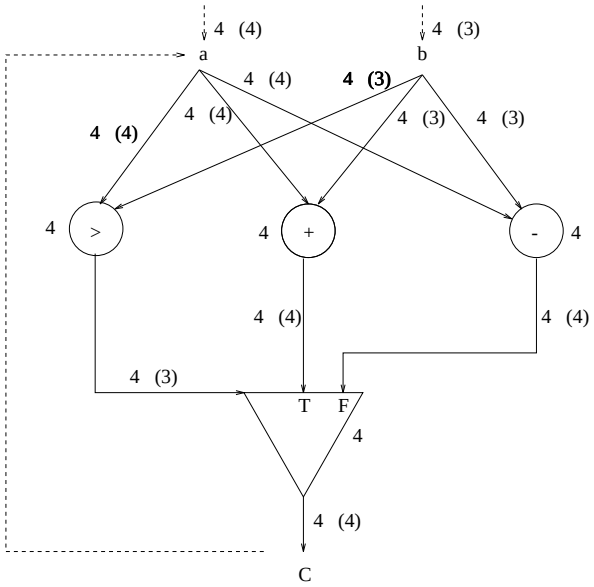

Figure 6: **DFG Annotated with Profile Data**

1. Generate a set of valid module bags.
2. **for each** valid module bag **do**
3. **begin**
4. Schedule the DFG using the module bag.
6. Estimate area, clock period and ASC.
7. **end**
8. Choose the schedule (and the corresponding operation bindings)
for which the area and delay estimates are within the
stated constraints and which minimizes the ASC.

Figure 7: **Scheduling and Performance Estimation**

length schedules and selects the one that meets both area and clock period constraints and has the least estimated ASC. Figure 7 shows the overall algorithm for the scheduling and performance estimation phase. Dutta et al. describe efficient algorithms for generation of valid module bags [12]. We use the force-directed scheduling algorithm proposed by Paulin and Knight [13]. Efficient procedures to estimate area and clock speed are described in literature [19, 20, 21, 9]. In this paper, we describe procedures for switching capacitance estimation only. Note that, at the end of scheduling, each DFG operation is bound to a library module instance. Hence, our power estimation process, described in the next section, assumes scheduled and operator-bound DFGs. Figure 8 shows a scheduled and operator-bound data flow graph.

Register Optimization Each data flow edge that crosses a control step boundary denotes a value that needs to be stored in a register. Such edges are called *carriers* of values. Register optimization phase groups carriers such that no two carriers in the same group are simultaneously active. Each group of carriers is then assigned to a register module whose bit-width is the maximum of the bit-widths of all carriers in the group. Various register optimization strategies have been proposed [30]. Our register optimization algorithm is based on a clique partitioning heuristic on the lines proposed by Tseng and Siewiorek [31]. Figure 9 shows the DFG following register optimization and binding.

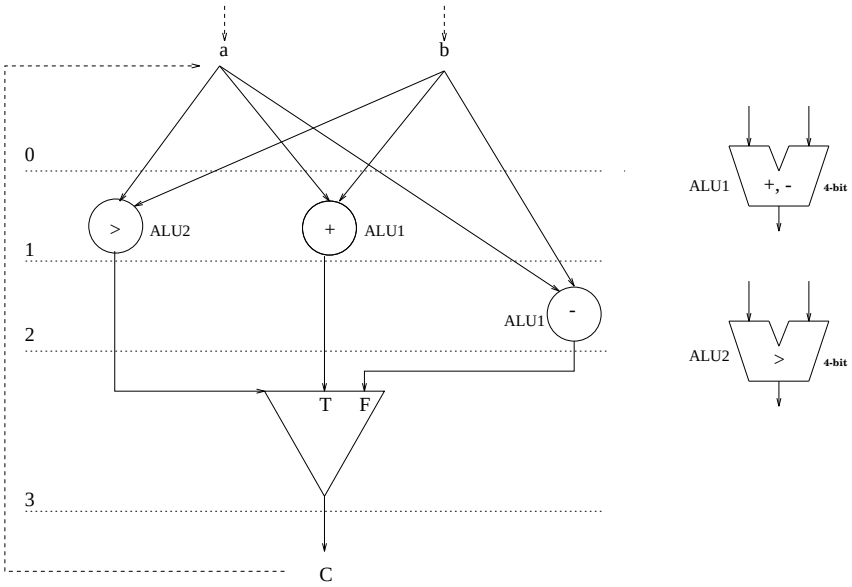

Figure 8: **Scheduled and Operator-Bound DFG and Partial Data Path**

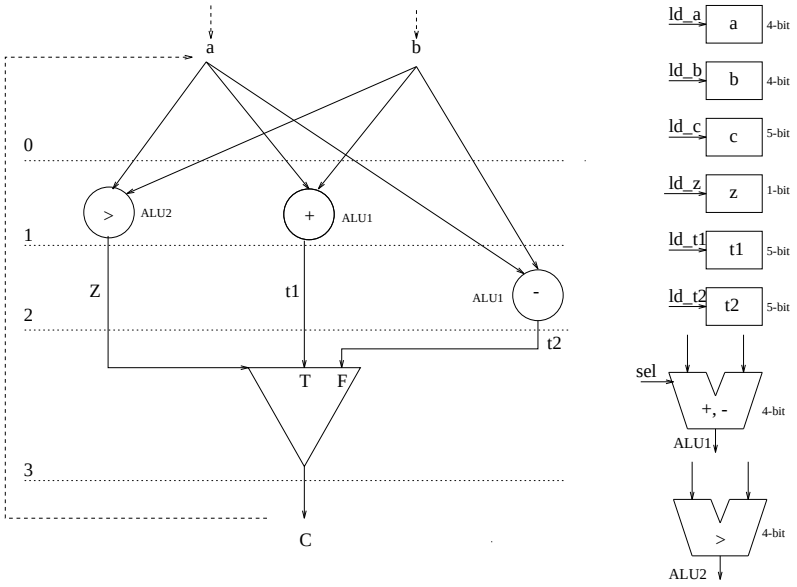

Figure 9: **DFG Following Register Optimiation and Partial Data Path**

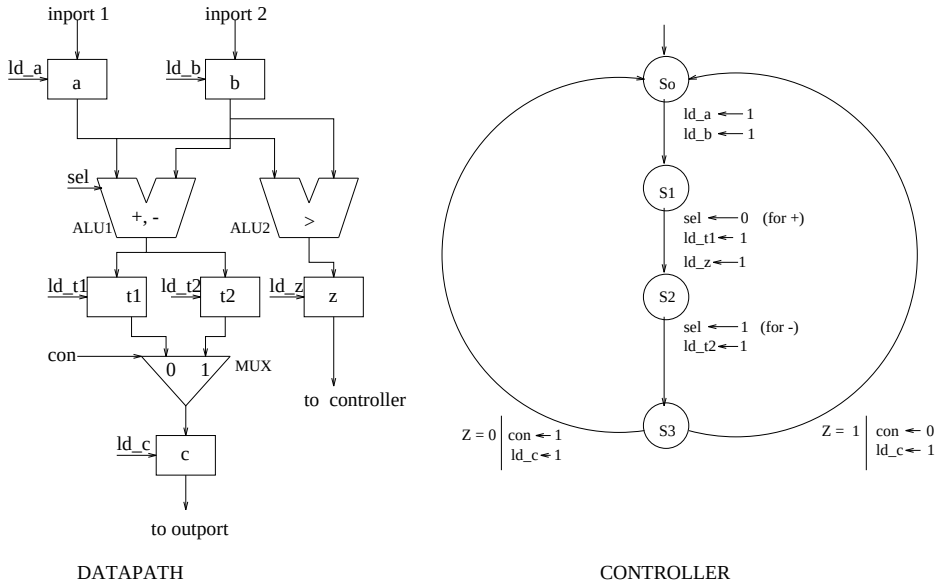

Figure 10: **Data Path and Controller after Interconnect Optimization**

Interconnect Optimization The goal of interconnect optimization is to assign an interconnect path to each value transfer in the DFG. Interconnect paths are formed using point-to-point wires, multiplexors and gated buses (buses whose drivers are enabled through transmission gates). Various cost functions and algorithms for interconnect optimization exist [32]. Our interconnect optimization algorithm is a min-max algorithm that prefers shared buses to multiplexor-based connections wherever possible. Figure 10 shows the data path and the controller generated following interconnect optimization for our illustrative example.

Controller Generation Following interconnect optimization, the data path is completely formed. Controller generator produces a finite state machine (FSM) description. The FSM accepts inputs in the form of status *flags* from the data path and produces *control signals* which would enable register transfers in the data path. Each control input of a control node in the DFG denotes a flag from the data path to the controller. DFG edges that cross control step boundaries denote state transitions in the FSM. Each

control step corresponds to at least one state in the FSM. In the presence of mutually-exclusive conditional branches, a control state may be mapped to multiple FSM states, one per each exclusive branch. Controller generation algorithms have been described in the literature [33]. Figure 10 shows the controller FSM for our example.

4 Estimation of Aggregate Switching Capacitance

Aggregate Switching Capacitance (ASC) is defined as the total capacitance switched in the design when simulated using the profiling stimuli.
ASC estimation takes place each time the DFG is scheduled using a module bag and partially bound. At this time, all operation nodes are bound to modules in the library. ASC for the entire design is given by

$$\text{ASC}_{design} = \text{ASC}_{dp} + \text{ASC}_{con} + \text{ASC}_{clock}$$

where ASC_{dp} , ASC_{con} and ASC_{clock} are the capacitances switched by the data path, controller and the system clock respectively.

4.1 Estimation of Data path ASC

The data path is composed of three types of components : (1) Register Units; (2) Operators; and (3) Interconnect. Thus the ASC_{dp} is given by

$$\text{ASC}_{dp} = \text{ASC}_{reg} + \text{ASC}_{op} + \text{ASC}_{i}$$

4.1.1 Estimation of ASC in Registers and Operators

The estimation of ASC_{reg} and ASC_{op} is as explained below. For any operator *op*, the capacitance switched is the product of its event activity and the ISC of a module to which the operator is bound. For any carrier c, the capacitance switched is the product of its event activity and the ISC of the register module with bit-width equal to the bit-width of the carrier. Thus ASC_{reg} and ASC_{op} are given by the equations,

$$\text{ASC}_{op} = \sum_{op} E_{op} \ * ISC_{bound(op)}$$

$$\text{ASC}_{reg} = \sum_{c} E_c * ISC_r$$

4.1.2 Estimation of ASC in Interconnect

The estimation of ASC_i is difficult due to the fact that the number and sizes of the interconnect units is known only after allocation and binding phases are carried out. Typically, during allocation phase, the hardware units for the carriers in the DFG are chosen by exploring various operator sharing and register sharing possibilities. The allocation phase is followed by the binding phase in which the possibility of interconnect sharing is explored. As the final interconnect configuration is dependent on the allocation and binding phases which are dependent on each other, predicting the final interconnect configuration is very difficult.

The strategy to estimate ASC_i is as follows: Assuming that in the final design, maximum operator sharing or maximum register sharing takes place , we can estimate an upper bound on the amount of interconnect introduced by each type of sharing. The sum of these two bounds is the upper bound on the amount of interconnect in the design.

The estimation of the upper bound on interconnect due to operator sharing is as follows: For each operator type (like *add*, *multiply* etc.,) assuming all the operations (say n) are scheduled in their ASAP timesteps, the maximum number of instances (say m) that are needed in the same control step can be computed. This is the minimum number of hardware units that are necessary for the operator type, in the entire design. Obviously this introduces maximum interconnect. Assuming binary operations, the maximum number of interconnect units introduced due to operator sharing [2] is 2 m, since a maximum of one multiplexor will be introduced at each input port of the operator. Assuming mux-based design, we estimate the capacitance switched as follows: The parameters of any multiplexor are: bitwidth and number of inputs. Each of these units has bitwidth equal to the maximum of the widths (say W_{max}) of all operator instances. The number of inputs is equal to n / m, assuming n operator instances are bound equally amongst m interconnect units. The event activity of each interconnect unit is equal to the sum of the event activities of all the operator instances E_{op_type}, divided by m. The capacitance switched in the interconnect introduced by the sharing of operator of type *op_type* is given by

$$\text{ASC}_{mux,op\_type} = m * (E_{op\_type}/m) * ISC(W_{max}, n/m, op\_type)$$

[2] Operator chaining is not considered.

$$= E_{op\_type} * ISC(W_{max}, n/m)$$

where $E_{op\_type} = \sum_{op} E_{op}$ and $ISC(width, number\_of\_inputs\ , type)$ is the ISC of a module whose type is *type* and has the parameters *width* and *number_of_inputs*. The capacitance switched in the interconnect introduced by the operator sharing is given by

$$\text{ASC}_{mux\_op\_share} = \sum_{op\_type} \text{ASC}_{mux,op\_type}$$

The upper bound on interconnect due to maximum register sharing alone is determined as follows: Let the maximum number of edges crossing any control step boundary be R_{min}. Then at least R_{min} number of register are needed in the entire design. The total number of edges (R_{max}) that cross all control step boundaries is the maximum number of register instances that are needed in the entire design. Thus the upper bound on the total number of interconnect units due to maximum register sharing alone is approximately , $N = R_{max}/R_{min}$. The event activity for any interconnect unit is approximately the sum of the event activities (say E_{reg}) of all the carriers divided by N. Assuming mux-based design, the width of any interconnect unit is the maximum of the widths of all the edges. The number of inputs is roughly R_{max} / R_{min}. Using these two parameters and the event activity, switching capacitance due to this interconnect can be estimated.

$$\text{ASC}_{mux\_reg\_share} = N * (E_{reg}/N) * ISC(W_{max}, N, Register)$$

$$= E_{reg} * ISC(W_{max}, N, Register))$$

The aggregate switching capacitance in the entire interconnect, ASC_i given by

$$\text{ASC}_i = w_i * (\text{ASC}_{mux\_op\_share} + \text{ASC}_{mux\_reg\_share})$$

where w_i is the weighting factor less than 1. We assume that $w_i = 0.5$ i.e only 50 % of operator sharing and 50 % of register sharing occurs in the final design.

4.2 Estimation of ASC in the Controller

PDSS implements the *controller* of a design as a PLA. As discussed in 3, ISC per clock step of a PLA with three parameters, *input size* ($\mathcal{I}$), *output size*($\mathcal{O}$) and the *number of states*($\mathcal{S}$), is readily available from the

PLA characterization table. Thus to estimate the ASC_{con} we need to estimate $\mathcal{I}, \mathcal{O}, \mathcal{S}$ of the controller and T_v, the total number of clock steps needed to process an input vector. T_v is assumed to be approximately equal to the number of states in the controller. Hence the total number of clock steps for the entire set of stimuli is equal to the number of stimuli N_v multiplied by T_v.
The aggregate switching capacitance in the controller ASC_{con} is computed as follows:

$$\text{ASC}_{con} = N_v * T_v * ISC(\mathcal{I}, \mathcal{O}, \mathcal{S})$$

where ISC($\mathcal{I}, \mathcal{O}, \mathcal{S}$) is the ISC of a PLA of with parameters $\mathcal{I}, \mathcal{O}, \mathcal{S}$. If we do not have an entry for $(\mathcal{I}, \mathcal{O}, \mathcal{S})$ in the PLA characterization table, then extrapolation/interpolation is carried out.

4.3 Estimation of ASC for the system clock

The system clock is used to clock all the components in data path and controller and hence it is loaded by large capacitance. As it is frequently switched, it consumes substantial amount of power. In data path, the clock goes to the storage units like registers and latches. Each type of storage unit in the module library is characterized for the typical capacitive loading on the clock lines for different bitwidths. Note that the capacitive loading by the controller is already taken care of in the PLA characterization. Thus we need to estimate the capacitance switched by the clock in the data path alone.

$$\text{ASC}_{clock} = N_v * T_c * \sum_c \mathcal{C}_c$$

where $\mathcal{C}_c$ is the capacitive loading on the clock lines of the register to which the carrier c is bound.

5 Results

In this section we present experimental results for six designs: a compression chip, a decompression chip, a sort and search chip (Find), a first-in first-out queue (FIFO), a traffic light controller (TLC) and a Shuffle Exchange Network [14]. Table 3 shows the specification and profiling data. Table 4 shows pertinent data obtained during the synthesis process. PDSS system is implemented in C++ on Sun Sparcstation I platforms.

Sl.	Design	LOC	DFG Nodes	DFG Edges	Profiling Stimuli	Profiling Time (s)
1.	TLC	72	27	123	18	2.8
2.	Compress	42	22	107	7	2.0
3.	Decompress	40	22	106	8	1.9
4.	Find	63	33	121	8	4.8
5.	FIFO	70	38	176	6	4.0
6.	Shuffle Xchg NW	450	31	2040	14	30.9

Table 3: **Specification and Profiling Data for Six Designs**

Sl.	Design	Number of Schedules	Schedule Length	Carriers	Transitions	Synthesis Time (s)
1.	TLC	2	17	18	27	1.5
2.	Compress	1	14	15	53	1.0
3.	Decompress	1	13	13	52	1.0
4.	FIFO	2	10	29	57	1.7
5.	Find	2	34	29	45	2.2
6.	Shuffle Xchg NW	4	13	214	98	24.0

Table 4: **Data During and After Scheduling**

Sl.	Design	Clock Period	Nodes	Transistors	Area (sq.mm.)	Cycles	Simulation Time (min)
1.	Decompress	200ns	2,803	6,059	10.3	259	1.04
2.	Compress	200ns	2,946	6,315	10.9	401	1.27
3.	Find	550ns	5,602	11,458	20.3	2,662	17.44
4.	FIFO	900ns	4,438	10,688	24.6	580	5.11
5.	TLC	200ns	1,938	4,769	6.9	760	3.16
6.	Shuffle Xchg	160ns	49,655	95,004	418.7	1,975	240

Table 5: **Synthesized Design Data**

Each register level design produced by PDSS is processed by the Lager IV silicon compiler [15] to generate mask layouts. The designs generated use a two phase non-overlapping clocking scheme. Although the designs are generated in a scalable CMOS technology, all results for this paper are obtained using 2 micron feature size. Switch level models are extracted from the layouts and simulated using the IRSIM switch level simulator. IRSIM uses an RC model for timing calculations. Table 5 shows the data of the final synthesized design.

Table 6 shows the estimated and actual ASC in the data path and controller of the six designs. The estimated ASC is computed during the scheduling phase of PDSS and actual ASC is determined during the switch level simulation of the synthesized designs. As shown in the table, the percentage error in ASC estimation for data path is in the range of 3.9% – 14.90% with the average deviation from the actual value being 8.0%. Similarly for controller the range is 2.2% – 13.8% with the average deviation being 9.9%. Table 7 shows the ASC values for the entire design, which is the sum of the ASC in data path, controller and system clock. The percentage error is in the range 9.8% – 14.4% and the average deviation is 11.9%. This shows excellent correlation between the estimated and actual values not only in the entire design but also in the datapath and controller seperately.

Sl.	Design	Data path			Controller		
		Estimated pF	Actual pF	% Devn.	Estimated pF	Actual pF	% Devn.
1.	TLC	9400.31	9826.92	4.30	32163.58	32910.90	2.2
2.	Compress	8174.54	7113.63	14.90	15020.66	16408.90	8.4
3.	Decompress	5498.05	4992.90	10.10	9437.49	8470.89	11.4
4.	FIFO	24950.29	27445.31	9.09	73041.69	82546.00	11.5
5.	Find	202913.69	215996.00	6.05	211097.55	187340.00	12.6
6.	Shuffle Xchg	545976.25	525207.00	3.95	256303.50	297400.03	13.8
Average Error				8.06			9.98

Table 6: **Comparison of the ASC in the Datapath and Controller**

Sl.No	Design	Total		
		Estimated pF	Actual pF	% Devn.
1.	TLC	45216.09	50132.90	9.8
2.	Compress	26654.46	23293.80	14.4
3.	Decompress	17685.64	15708.22	13.7
4.	FIFO	104210.31	116278.00	10.3
5.	Find	446851.69	399834.00	11.7
6.	Shuffle	1227679.75	1388943.00	11.6
Average Error				11.9

Table 7: **Comparison of ASC for the Entire Design**

6 Conclusions

In this paper we presented a new, profile based approach for the estimation of switching capacitance at the behavioral level and used these estimates to synthesize a low power design. This approach attempts to make realistic estimates of switched capacitance by taking into account the effect of input stimuli. The results are within an accuracy of 12% of the actual switching capacitance measured at the switch level implementation of the design.

References

[1] C. M. Huizer, "Power Dissipation analysis of CMOS VLSI circuits by means of switch level simulation", IEEE European Solid State Ciruits Conference, Grenoble, France, pp. 61-64,1990.

[2] R. Burch, F. Najm, P. Yang, and T. Trick, " McPOWER: A Monte Carlo approach to power estimation ," *IEEE/ACM International Conference on Computer-Aided Design,* Santa Clara, CA, pp. 90-97, November 8-12, 1992.

[3] M. Xakellis and F. Najm, "Statistical Estimation of the Switching Activity in Digital Circuits," *31st ACM/IEEE Design Automation Conference,* San Diego, CA, pp. 728-733, 1994.

[4] Anantha P. Chandrakasan et al., "HYPER-LP: A System for Power Minimization UsingArchitectural Transformations", Proc. ICCAD, pp.300-303, November 1992.

[5] Anantha P. Chandrakasan et al., "Optimizing Power Using Transformations", IEEE Transactions on Computer Aided Design, pp. 12-31, January, 1995.

[6] Renu Mehra and Jan M. Rabaey, "Behavioral Level Power Estimation and Exploration", Proceedings of the International Workship on Low Power Design, pp. 165-170, April 1994.

[7] Scott R. Powell and Paul M. Chau, "A Model for Estimating Power Dissipation in a Class of DSP VLSI Chips", IEEE Transactions on Circuits and Systems, Vol. 38, No. 6, June 1991.

[8] Paul Landman and Jan Rabaey, "Power Estimation for High Level Synthesis", Proc. of EDAC-EUROASIC, pp 361-366, February 1993.

[9] P. K. Jha and N. D. Dutt, "Rapid Estimation for Parameterized Components in High-Level Synthesis", IEEE Transactions on VLSI Systems, vol. 1, no.3, pp. 296-303, September 1993.

[10] Ravi Kalyanaraman, "Behavioral Test Generation for VHDL Programs", MS Thesis, Department of Electrical and Computer Engineering, University of Cincinnati, September 1993.

[11] Darrel Ince, "Software Testing", in John McDermid (ed.) *Software Engineer's Reference Book*, Butterworth-Heinemann Ltd., 1991.

[12] Rajiv Dutta, Jay Roy, and Ranga Vemuri, "Distributed Design-Space Exploration for High-Level Synthesis Systems," 29th Design Automation Conference, pp. 644–650, June 1992.

[13] P.G. Paulin and J.P. Knight, "Force-Directed Scheduling in Automatic Data Path Synthesis," 24th Design Automation Conference, pp. 195–202, June 1987.

[14] "Novel IC Shuffles Parallel Processing Data", Electronic Products, pp. 42-50, August 1, 1986.

[15] Rajeev Jain et al., "An Integrated CAD System for Algorithm-Specific IC Design ", *IEEE Transactions on Computer Aided design*, Vol 10, No. 4, April 1991.

[16] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits", IEEE Trans. CAD, vol. 12, no. 2, pp. 310-323, February 1993.

[17] F.N.Najm,"A Survey of Power Estimation Techniques in VLSI circuits" IEEE Trans. VLSI Systems, vol. 2, no. 4, pp. 446-455, January 1995.

[18] G.I. Stamoulis and I.N. Hajj, "Improved techniques for probabilistic simulation including signal correlation effects," *30th ACM/IEEE Design Automation Conference,* pp. 379-383, 1993.

[19] M. J. Mlinar, "Control Path/Data Path Tradeoffs in VLSI Design," Ph.D. Dissertation, Department of Electrical Engineering-Systems, University of Southern California, CA, June 1991.

[20] F.J. Kurdahi, "Area Estimation of VLSI Circuits," Ph.D. Dissertation, Dept. of Electrical Engineering, University of Southern California, CA, 1987.

[21] W. H. Wolf, "How to Build a Hardware Description and Measurement System on an Object Oriented Programming Language", IEEE Transactions on CAD, vol. 8, no. 3, pp. 288-301, 1989.

[22] J. Monteiro, S. Devadas and B. Lin, "A Methodology for Efficient Estimation of Switching Activity in Sequential Logic Circuits", 31st Design Automation Conference, pp. 12-17, 1994.

[23] S. M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits", *IEEE J. of Solid-State Circuits,* vol. 21, no. 5, pp. 889-891, October 1986.

[24] G.Y. Yocoub and W.H. Ku, "An accurate simulation technique for short-circuit power dissipation based on current component isolation ," *IEEE International Symposium on Circuits and Systems,* pp. 1157-1161, 1989.

[25] A-C. Deng, Y-C.Shiau, and K-H. Loh, " Time domain current waveform simulation of CMOS circuits," *IEEE International Conference on Computer-Aided Design,* Santa Clara, Ca, pp.208-211, Nov. 7-10, 1988.

[26] R. Tjarnstrom, "Power dissipation estimate by switch level simulation," *IEEE International Symposium on Circuits and Systems*, pp. 881-884, May 1989.

[27] T.H. Krodel, "Power-Play - fast dynamic power estimation based on logic simulation," *IEEE International Conference on Computer Design*, pp.96-100, October 1991.

[28] L.Benini, M.Favalli, P. Olivo, and B. Ricco, "A novel approach to cost-effective estimate of power disipation in CMOS ICs, *European Design Automation Conference* , pp. 354-360, 1993.

[29] F. Dresig, Ph. Lanches, O. Rettig, and U.G. Baitinger, "Simulation and reduction of CMOS power dissipation at logic level," *European Design Automation Conference,* pp. 341-346, 1993.

[30] F. Kurdahi and A. Parker, "REAL : A Program for REgister ALlocation," 24th Design Automation Conference, pp. 210-215, 1987.

[31] C. Tseng and D. P. Siewiorek, "Facet : A Procedure for the Automated Synthesis of Digital Systems," 20th ACM/IEEEDesign Automation Conference, pp. 490-496, 1983.

[32] B.M. Pangrle, "Splicer : A Heuristic Approach to Connectivity Binding," 25th ACM/IEEE Design Automation Conference, pp. 536-541, Jun. 1988.

[33] A. W. Nagle, R. Cloutier and A. C. Parker, "Synthesis of Hardware for the Control of Digital Systems", IEEE Transactions on CAD, vol. 1, no. 4, October 1982.

[34] M. A. Cirit, "Estimating Dynamic Power Consumption of CMOS Circuits", Proc. ICCAD, pp. 534-537, November 1987.

[35] R. Burch, F. Najm, P. Yang, and D. Hocevar, "Pattern-independent current estimation for reliability analysis of CMOS circuits," *25th ACM/IEEE Design Automation Conference*, Anaheim, CA, pp. 294-299, June 12-15, 1988.

[36] C-Y. Tsui, M. Pedram, A.M. Despain, "Efficient estimation of dynamic power consumption under a real delay model," *IEEE Internations Conference on Computer-Aided Design*, Santa Clara, CA, pp. 224-228, November 7-11, 1993.