High Level Synthesis of Data Driven ASICS *

Baiju Patel
T.J. Watson Research Center, IBM
P.O. Box 704, Room H2D20
Yorktown Heights, NY 10598

Dhiraj K. Pradhan and Israel Koren
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Abstract

A novel approach to high level synthesis of ASICs based on a data driven execution model is presented. The synthesis procedure is directed at producing highly parallel ASICs providing high throughput through pipelining. The major benefits of our approach are its potential for higher speed, ease of design, ease of verification and testing.

I Introduction

Proposed here is an architecture and synthesis procedure for design of high performance VLSI circuits for a specific application, commonly referred to as application specific IC or ASIC. A key objective of our design is to achieve high performance by first exploiting the parallelism inherent in the application and secondly, providing pipelining capabilities. Other potential benefits of our approach include simple partitionability, ease of testability and design verification.

Specifically, in our approach, the application is first described in SISAL, a data flow language. This SISAL program is then translated to a data flow graph (DFG) using an optimizing compiler developed at Lawrence Livermore National Lab. This DFG is then "directly" mapped onto VLSI where each node in the DFG constitutes a cell in VLSI. The operation of these VLSI cells follows the data-driven execution model; i.e., each cell executes its task as soon as all the necessary operands are available at the inputs. This "direct" mapping of the DFG onto VLSI, coupled with the data driven execution, make it possible to exploit the inherent parallelism in the application itself. Therefore, the designer is not required to identify the parallelism in the application. Storage buffers are added to the arcs in order to support pipelined operation of the ASIC. Thus, providing high levels of concurrency and pipelining are the two major goals in our design.

A novel area minimization algorithm is developed to select implementations from the cell library to minimize the overall area of the ASIC for given performance constraints. Also developed is a buffer allocation algorithm to balance the DFG for our data driven architecture. The area minimization and buffer allocation steps are carried out to meet specified area and performance requirements. We show that by the very nature of the ASIC generated using our approach, it is simple to test the control section of the ASIC. Simple testability features are incorporated in the data section to facilitate the functional test generation. The technique is extended to the synthesis of multichip ASICS. Synthesis tools incorporating all these features are being implemented.

II Data Driven Architecture

Conventionally, a control driven architecture has been used for implementing ASICs. The control driven architecture is partitioned into two major components:(i) Control and (ii) Data paths. The data paths are composed of a network of registers, functional units, multiplexers and buses[4]. The data path carries out all the operations and also stores intermediate results. The central control explicitly orchestrates all the actions on the data paths. The control is designed as a finite state machine (FSM). This architecture is geared towards compact designs with high degree of sharing of resources leading to an highly area efficient ASIC. The low cost also results in lesser performance.

The data driven architecture is envisioned based on two important observations: (i) The data flow graph for the application represents all the inherent parallelism in the algorithm and (ii) If storage buffers are placed on the arcs, it can be executed in pipeline fashion. Of course, such architecture can follow a synchronous execution model used by control driven architectures whereby each operator is scheduled to execute at a predetermined time. However, this will inhibit the ASIC to efficiently realize applications which include conditional statements and loops[6]. In order to overcome this shortcoming of synchronous execution, we elect to use the data driven execution mode of operation.

The data driven execution is implemented based on well known handshaking mechanism. A block diagram of a representative node and buffer is shown in the Figure 4. We use uniform handshake for all the nodes in order to keep the design process simple. Moreover, the same handshake protocol is used for data transfer within the chip and between chips. This feature overcomes complicated synchronization problems that arise in control driven architectures.

The area of the handshake control is of the order of 10% for add and subtract node. The overhead is even smaller for large nodes such as multiplication and division. The total delay added to the clock period due to the handshake control is that of approximately three to four gate delays which is fairly small.

In summary, our architecture uses one functional module for every node of the data flow graph to exploit all the parallelism available in the DFG. A buffer is placed on every arc of the DFG to facilitate pipelining. The nodes are executed based on the data driven execution model in order to minimize the pipeline period and latency in case of data dependent execution time. At first glance, it may seem that it is too expensive to use one functional module for every node and it may be true in many cases. Therefore, we have developed algorithms which will trade performance for cost.

III Design Process

The complete procedure for designing data-driven ASICs is shown in Figure 1. These steps are outlined in what follows. The details may be found in [8]

SISAL to DFG Translation: The application program is specified in SISAL [5], which is then translated using an optimizing compiler to a DFG.

Area Estimation/Minimization: Different operands at a multi-input node may arrive at different time instances by traversing different paths. Since the operand that arrives earlier has to wait for the other operands to arrive, the delay of the nodes on the shorter paths can be increased without affecting the overall performance. This, in turn, will reduce the area of the ASIC.

A mathematical programming formulation was developed

*This work was supported in part by SRC under contract 90-DJ-125 to obtain the lower bound on the area of the ASIC. If the area lower bound is unacceptably high, then either the application program must be modified or the performance requirement must be relaxed. We have developed two algorithms to assign implementations from the cell library to the nodes: a greedy algorithm to obtain a "good" solution, and a branch and bound algorithm to obtain an "optimal" solution.

Buffer Allocation: If a DFG with non-uniform path lengths is directly mapped onto VLSI, it may not be optimally pipelinable. In such an event, buffers may be added to shorter paths of the DFG. The buffer allocation problem for this architecture is more difficult than the similar problem for synchronous pipelines [3] and static data-flow computers [2] because data-driven ASICs allow variable execution time for nodes and variable delay for buffers. This problem has been mapped to a quadratic programming problem that can be solved using well-known methods.

Testing: Many of the existing test generation techniques (e.g., [1]) may be used for our architecture. However, a general purpose test generation methodology may not be capable of identifying and exploiting the architecture specific properties of the data driven ASICs such as hierarchy, small number of pre-designed cells, and distributed and uniform control. We have developed a test generation methodology which uses architecture specific information on our architecture to facilitate their testing. First, a functional fault model and test generation techniques based on it are developed. Then, these techniques are validated by demonstrating that stuck-at-faults in the current implementation are tested.

Layout Generation: The final layout is generated using OCT tools [9] developed at the University of California at Berkeley. A library of behavioral descriptions of the different node implementations has been prepared. Then, the entire ASIC is synthesized from this library of behavioral descriptions using standard cell based generators.

IV Area Estimation/Minimization

The need for area estimation/minimization can be illustrated through the example in Figure 2. Let the *multiply* (MULT) node be a sequential multiplier that takes 16 cycles for the computation. Let the *add* (ADD1 and ADD2) nodes be parallel 16-bit adders that take one cycle to execute. Therefore, for the initial mapping, the result of *MULT* is available 15 cycles later than the result of *ADD1*. Since *ADD2* cannot execute until both operands are available, the result of *ADD1* has to wait for 15 cycles after it has been computed. However, an alternate bit-serial implementation (with execution time of 16 cycles) may be used for node *ADD1* without affecting the overall performance of the ASIC. Moreover, the area of the interconnections for *ADD1* would be reduced as well. A parallel adder should still be used for *ADD2* because any slower implementation will increase the length of the critical path and thus increase the latency.

Area Estimation: In the following we formulate the problem of finding a minimum area ASIC for a given DFG, such that it satisfies the performance requirements on latency and pipeline period. Let

- $V = (n_1, \ldots, n_l)$: set of the nodes of the DFG
- $E = (e_1, \ldots, e_m)$: set of the edges of the DFG
- $T_n(n_i)$: the execution time for node n_i
- $A_n(n_i)$: the area of node n_i
- P: pipeline period of the ASIC
- L: latency of the ASIC

Let ϕ be a path from an input of the DFG to a node or a primary output. Then, in order to satisfy the latency requirements, the length of all the paths from the inputs to the outputs of the DFG must not be larger than the desired latency. Hence

$$\sum_{\forall n_i \in \phi} T_n(n_i) - L \leq 0 \tag{1}$$

This *latency constraint* must hold for all the paths between the inputs and outputs of the DFG.

Since the overall pipeline period is determined by the largest delay of all the nodes in the DFG, the delay of each node must not be larger than P. Therefore, the *pipeline period constraint* is:

$$T_n(n_i) - P \leq 0 \quad \forall n_i \in V \tag{2}$$

For a multi-input node to carry out a computation correctly, all the inputs must be available at the execution time. Therefore, any two paths from primary inputs to two inputs of any node must have the same length. Let two such paths be ϕ_l and ϕ_k then the *equal path length constraint* is

$$\sum_{\forall n_i \in \phi_k} T_n(n_i) - \sum_{\forall n_j \in \phi_l} T_n(n_j) = 0 \tag{3}$$

Finally, the *area/delay relationship* for a node must be satisfied as well which may be different for different types of nodes and may be based on available design methodologies.

Given the performance requirements in terms of L and P, the objective is to determine the values of $A_n(n_i)$ and $T_n(n_i)$ such that the overall area ($\sum_{\forall n_i \in V} A_n(n_i)$) of the ASIC is minimized.

Even though the mathematical programming solution to the synthesis process provides an optimal area solution for the data driven ASICs, it may not always yield a practical solution. Therefore, we developed a *Greedy Heuristics* and a *Branch and Bound* algorithm to solve the area minimization problem for the available implementations of the nodes.

A Greedy Algorithm

In this algorithm, a node for area reduction using the following guidelines is selected. First, it may be noted that a node cannot start its execution before all its required inputs are available. Therefore, the earliest time it can execute is the latest time instance at which the results from all the predecessor nodes are available. Similarly, the node must complete its execution before the latest time instance at which its output must be available to all the successor nodes. These two time instances are denoted by *asap* (as soon as possible) and *alap* (as late as possible) respectively. The *asap* and *alap* times, also referred to as *slack* and *surplus* times, have been used for several scheduling problems (e.g., [4]). We define the *freedom* of a node as the difference between the *alap* and *asap* times. The freedom is the largest amount by which the node execution time can be increased without increasing the overall latency. Obviously, the freedom of a node on the critical path is zero. Amongst the nodes with non-zero freedom, that node which results in maximum area reduction is selected. Whenever there is more than one such candidate node, the one with the smallest freedom is selected. This node is then replaced by a smaller implementation. This process is repeated until no new candidate node can be found for area reduction. If the final area of the resultant ASIC is still unacceptably large, the length of the critical path (latency) is increased and the process is repeated. The greedy algorithm based on the above is summarized as follows.

Algorithm: Area Minimization

1. Compute the freedom for each node.

2. Let S be the set of candidate nodes for which there are implementations such that when replaced, the increase in the delay is not larger than their freedom.

   If S is empty then exit.

3. Compute the *area savings* for each node v in S.

4. Let $S' = \{v \mid v \in S$ and v has maximum area savings$\}$

5. Choose node v from S' with smallest freedom.

6. Replace current implementation by a smaller one.

7. Go to step 1.

Branch and Bound Algorithm

As seen above, it is possible to replace the implementation of a node by a smaller one only if the resulting increase in the execution time is smaller than its freedom. Therefore, in this algorithm, first a lower bound for the area of the DFG is evaluated by choosing the smallest implementation possible (within the above constraint). The area lower bound is used during both the branching and bounding steps. At the branching step, out of all possible nodes that can be replaced by smaller implementations, the node whose replacement results in the smallest lower bound is selected. A tie in this selection is broken in favor of the node with the smallest freedom. If there are more than one such node with the smallest freedom, then the node with the largest area savings is selected. Bounding takes place when the current smallest solution is larger than the area lower bound.

V Examples

To demonstrate the area minimization algorithms we use an example taken from [7], shown in Figure 3, where "S" and "M" nodes are *Switch* and *Merge* nodes, respectively. The add and subtract nodes were designed using 2μ technology. The area and execution time for different implementations of these nodes are shown in Table 1. The "bits" indicate the number of bits that are operated simultaneously, i.e., 8 bits indicate that 16 bit addition is performed by adding 8 bits at a time in two clock cycles. For latency of 8 and pipeline period of 2, an 8-bit adder implementation (with a delay of two clock cycles) is used for *ADD3, ADD4, ADD5, SUB2, SUB4* and *SUB5*, and a 4-bit adder (with a delay of four clock cycles) for *SUB1*. The rest of the nodes are 16-bit adder implementations. Consequently, the total area for the initial mapping is reduced from 15.32 mm^2 to 13.69 mm^2, while keeping the overall execution time unchanged (16 clock cycles). Figure 5 shows the area lower bound and the different areas required using the greedy and branch and bound algorithms for the example of Figure 3. In all the examples that we have examined (including the above one), the minimized area obtained using the greedy algorithm was no more than 7% larger than the optimal area obtained using the branch and bound algorithm. The CPU time required for area minimization using both the greedy and branch and bound algorithms is shown in Table 2 where run time is measured in seconds on a Microvax II.

Therefore, we suggest to first generate, using the greedy algorithm, an initial set of solutions with different areas and latencies. Using these as input to the branch and bound algorithm, an optimal solution can be obtained. A layout obtained using this approach is shown in Figure 6.

VI Conclusions

An innovative approach to the design of ASICs has been presented in this paper. The designed ASICs operate in a data-driven mode that supports fine grain parallelism and pipelining. The developed CAD tool includes a compiler for translating the application specified in SISAL to a data-flow graph. This compiler also provides estimates for the performance of the ASIC. In the next step the area of the ASIC is minimized and then the final layout is generated. Examples illustrating the various steps in the design of the ASIC have been presented.

Our preliminary experiments show that a DFG with about 50 to 100 add/subtract operators can easily be implemented on a 1 cm^2 chip. Also, the timing analysis using Crystal and Spice shows that these ASICs can be operated at 10MHz to 20MHz clock rate.

time(clocks)	1	2	4	8
bits	16	8	4	2
area(mm^2)	0.957	0.750	0.577	.500

Table 1: The area and execution time for different implementations of a 16 bit adder.

latency	8	10	12	14	16	18	20
greedy	1	2	2.8	3.6	4.7	6.7	7.0
BB	8	480	732	4748	1745	4699	14470

Table 2: The run time (in seconds) for greedy, and branch and bound (BB) algorithms for the example of Figure 3.

References

[1] Agrawal, V., Cheng, K., and Agrawal, P. Contest: A concurrent test generator for sequential circuits. In *Proc. of 25th Design Automation Conference* (June 1988), ACM/IEEE, pp. 84–89.

[2] Gao, G. Algorithmic aspects of balancing techniques for pipelined data flow code generation. *Journal of Parallel and Distributed Computing 1*, 6 (Feb. 1989), 39–61.

[3] Leiserson, C., and Saxe, J. Optimizing synchronous systems. *J. of VLSI and Computer Systems 1*, 1 (1983), 41–67.

[4] McFarland, M., Parker, A., and Camposano, R. Tutorial on high-level synthesis. In *Proc. of 25th Design Automation Conference* (1988), pp. 330–336.

[5] McGraw, J. R., et al. SISAL: Streams and iterations in a single assignment language: Reference manual version 1.2. Manual M-146, Rev. 1, Lawrence Livermore National Laboratory, Livermore, CA, Mar. 1985.

[6] Mendelson, B., Patel, B., and Koren, I. Designing special-purpose co-processors using the data flow paradigm. In *Advanced Topics in Data-Flow Computing* (1990), J.-L. Gaudiot and L. Bic, Eds., Prentice-Hall.

[7] Parker, A., et al. MAHA: A program for datapath synthesis. In *Proc. of 23rd Design Automation Conference* (1986), pp. 461–466.

[8] Patel, B. *High-Level Synthesis of Data Driven ASICs*. PhD thesis, ECE Dept. University of Massachusetts, Amherst, 1990.

[9] Spickelmier, R., Ed. *Oct Tools Distribution 3.0.* University of California, Berkeley, Mar. 1989.

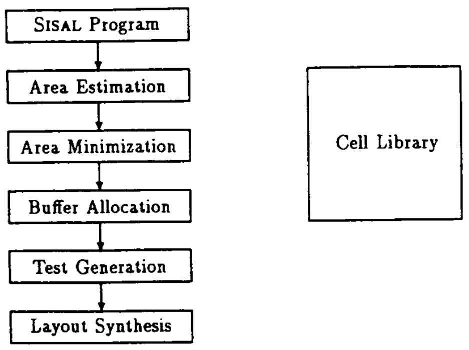

Figure 1: The design process.

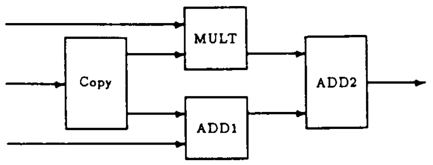

Figure 2: A simple DFG.

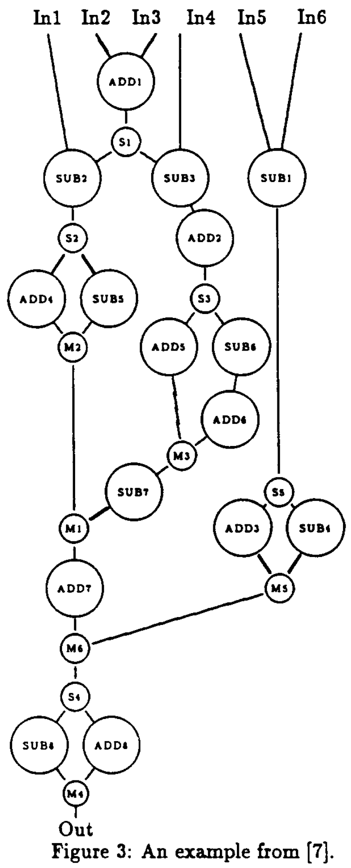

Figure 3: An example from [7].

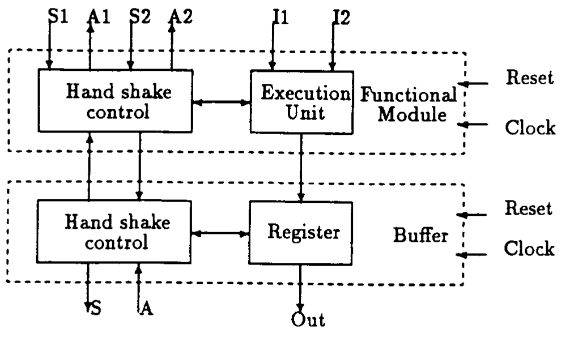

Figure 4: The block diagram of a node and a buffer.

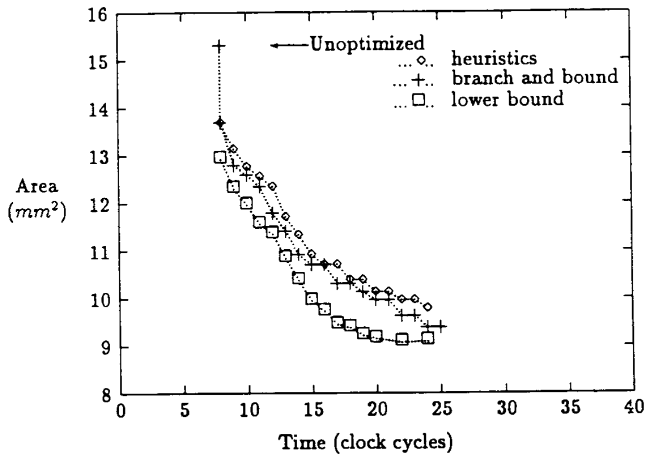

Figure 5: The minimum area required using greedy and branch and bound algorithms and the lower bound on area for the example of Figure 3.

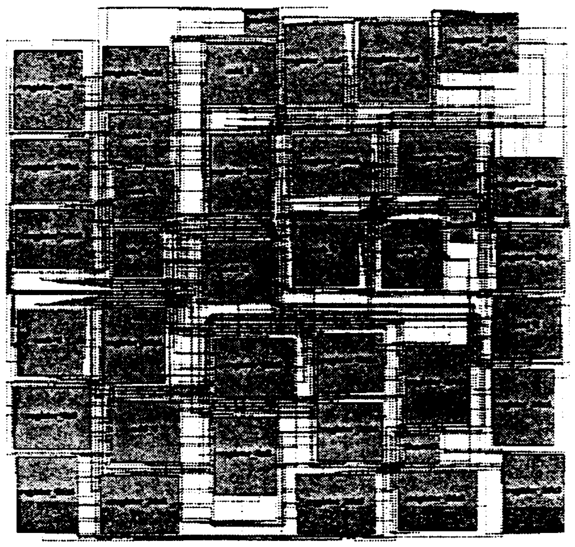

Figure 6: Final layout for the DFG of Figure 3