Improved Merging of Datapath Operators using Information Content and Required Precision Analysis

Anmol Mathur Sanjeev Saluja
Cadence Design Systems
555 River Oaks Parkway
San Jose, CA 95134

Abstract : *We introduce the notions of required precision and information content of datapath signals and use them to define functionally safe transformations on data flow graphs. These transformations reduce widths of datapath operators and enhance their mergeability. Using efficient algorithms to compute required precision and information content of signals, we define a new algorithm for partitioning a data flow graph consisting of datapath operators into mergeable clusters. Experimental results indicate that use of our clustering algorithm for operator merging based synthesis of datapath intensive designs, can lead to significant improvement in the delay and area of the implementation.*

1. INTRODUCTION

The number and complexity of datapath operations implemented in systems on chips has increased considerably over the years. This is especially true in chips for graphics, communication and multimedia processing applications, which have highly parallel implementation of signal processing algorithms such as fast fourier transforms, finite impulse response filters and opther DSP algorithms. While techniques for high performance synthesis of individual datapath operators like adders, multipliers, shifters are well known [3]; such datapath intensive RTL designs require synthesis techniques which yield optimized implementations of groups of datapath operators instead of individual operators.

One such useful technique is *operator merging* i.e. *clustering* of multiple datapath operators so that they can be synthesized together as a unit. In particular, designers and researchers have explored synthesizing a cluster of datapath operators as a sum of addends using carry-save adders and Wallace trees [2][4][5][6][7]. For example, synthesis of the sum of product expression *a*b+c*d* using traditional synthesis requires 2 multipliers and an adder. Such an implementation has 2 carry-propagate adders on any input-to-output path. Operator merging can implement such an expression using only one carry-propagate adder by reducing the partial products of the multipliers in a single carrysave reduction tree(CSA-tree). In [2], an operator merging based datapath synthesis algorithm has been proposed, which first partitions a data flow graph into clusters of datapath operators and then synthesizes each cluster using CSA-tree i.e. the combination of a reduction tree of carry-save adders and a final adder. The results of [2] demonstrate the effectiveness of operator merging in improving performance of netlists for datapath intensive designs. The later results of [4][5] focus on optimal implementation of the second step of the algorithm of [2] i.e. implementation of a sum of addends using carry-save adders and bit-oriented Wallace trees and they further support the usefulness of operator merging.

In this paper, we consider another approach to maximize the benefits of operator merging, which focuses on maximizing mergeability of operators so that larger and fewer clusters are created. The motivation for this approach comes from the observation that implementation of each cluster representing sum of addends implies the delay and area of a final carry-propagate adder; so partitioning a datapath computation into larger number of smaller clusters could mean more timing delay and area of the resulting netlist. Hence, increased merging implies reduction in the number of carry-propagate adders and consequently reduced critical path delay. Our experimental results mentioned in Section 7 validate the above intuition. Note that the above mentioned approach can be viewed as an attempt to improve the implementation of the first step of the algorithm of [2], i.e. partitioning a data flow graph into clusters.

A seemingly stronger motivation to consider the problem of maximal merging of datapath operators comes from the fact that the techniques that we use for partitioning data flow graphs into clusters, give a more general method of analyzing and optimizing data flow graphs consisting of datapath operators. In particular, they allow to safely reduce the bitwidths of datapath operators used in the design, which not only implies that the first pass of synthesis would generate faster and smaller netlists and but also that the gate level logic optimization step needs to work less to meet the timing and area constraints. Further, our method of partitioning a data flow graph into maximal mergeable clusters, also gives a method for safe partitioning of data flow graph, which can be used in problem scenarios other than operator merging e.g. rebalancing of computation graphs consisting of associative operators.

The main contributions of this paper are:

1. We define the notions of *required precision* and *information content* of a signal[1] which measure the essential bitwidth of the signal. We use these measures to define safe i.e. functionality preserving transformations on the data flow graphs (DFGs) which result in two benefits : the transformed graph has potentially smaller widths of datapath operators and has potentially greater mergeability of datapath operators.

2. We provide a characterization of safe clustering of DFGs in terms of required precision and information content of signals, that is applicable to DFGs which have both *signed* and *unsigned* extensions of signals.

3. We present efficient algorithms for computation of required precision and upper bounds on information content and the related DFG transformations. We use these algorithms to define a new iterative algorithm for partitioning a graph into maximal safe clusters.

[1] We use *signal* (or *datapath signal*) to mean bitvector representing value of any input, output or intermediate results in the data-flow graph.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

We note here that notions of required precision, information content, the related transformations and partitioning algorithm are applicable even to DFGs which have datapath operators besides addition, subtraction, unary minus and multiplication e.g. comparators and shifters. However, for the sake of clarity we restrict the discussion to $+$, $-$ and $\times$ operations in this paper.

The optimization algorithm of [2] uses the notion of *leakage of bits* to determine if operators at two adjacent nodes in data-flow graph can be merged. It turns out that the use of leakage of bits for deciding mergeability leads to an extent to similar mergeability boundaries as usage of analysis based only on required precision. However it turns out that, our clustering algorithm, which uses a combined analysis based on values of both required precision and information content, creates larger and fewer clusters and is a more elegant method of identifying mergeability bottlenecks in a data flow graph. Further, DFG transformations based on required precision and information content, give the additional benefit of reducing widths of datapath operators. [8] uses a notion similar to information content of a signal in the context of converting a floating point representation of a design to a fixed point representation.

The paper is organized as follows. Preliminary definitions and notations including definition and interpretation of a DFG, which are used throughout the paper, are given in Section 2. In Section 3, we use an example to illustrate the application of operator merging on a DFG and point out the essential characteristics of a cluster. Section 4 formally introduces the notion of required precision and shows how it can be used to transform a DFG and enhance the mergeability. Section 5 gives an analogous presentation for the notion of information content . Section 6 defines a new iterative algorithm for partitioning a DFG into maximal clusters, each of which is synthesizable as a sum of addends. In Section 7, we present preliminary results of our experiments, which compare the performance of netlists obtained on some datapath intensive test designs with and without our new clustering algorithm.

2. PRELIMINARIES

In this section, we give some definitions and notations which are used in the paper. In particular, we describe the DFG model that we have used as a representation of designs with datapath operators.

DEFINITION 2.1. *A* width extension *(or simply* extension*) of a signal is the padding of multiple copies of a fixed bit to the left of the signal to obtain a new signal of larger bitwidth. If the padding is done with bit 0, the extension is said to be* unsigned*. If it is done with the most significant bit of the original signal, the extension is said to be* signed*.*

For example, 00011 and 11111 are obtained from the two bit signal 11 by a five bit unsigned and five bit signed extension respectively.

2.1 Data Flow Graphs

A data flow graph (DFG) with datapath operators is a directed acyclic connected graph where nodes represent inputs, outputs and datapath operations. Edges represent the flow of data between operations. The interface of an edge with its source or destination node is referred to as a *port*. A port is an *input* (or *output*) port if it represents interface of an edge with its destination (resp. source) node. Each input (or output) node has one output (resp. input) port; each operator node has one output port and one or two input ports depending on whether the datapath operator on the node is unary or binary. The following quantities are defined for the nodes and edges in a DFG:

- Each operator node N has a *width* value $w(N)$, which is a positive integer. For an input (or output) node it represents, the bitwidth of the input (resp. output) signal represented by the node. For an operator node, it represents the number of bits used to represent the operands and/or result of the operation labeling the node.
- Each edge e has a *width* value $w(e)$, which is a positive integer. For an edge, the width represents the number of least significant bits of the result of the operation at the source node, which are used as input by the operation at the destination node of the edge.

  Each edge e is labeled with a binary attribute $t(e)$ called the *signedness* of the edge. The signedness is either *signed* or *unsigned*. We also use the binary bits $\{0, 1\}$ to represent the signedness types "unsigned" and "signed" respectively.

2.2 Interpretation of widths and signedness

Let N_1 and N_2 be the source and destination nodes of an edge e. Let their widths be $w(N_1)$, $w(N_2)$ and $w(e)$ respectively. If $w(e) \leq w(N_1)$, then signal defined by $w(e)$-many least significant bits of the result of N_1, is said to be *carried* by e. If $w(e) > w(N_1)$, then e carries a signal which is obtained by extending the result of N_1 to $w(e)$ width. The type of extension is determined by the signedness of the e. Similarly if $w(N_2) \leq w(e)$, signal defined by $w(N_2)$-many least significant bits of the signal carried by e is used as input operand by the operator at the destination node. If $w(N_2) > w(e)$ and implementation of the operator at N_2 requires an extension of its operand, then a $w(N_2)$ bit extension (whose signedness is determined by signedness of e) of the signal carried by e is used as the input operand.

3. MERGEABILITY OF DATAPATH OPERATORS

We illustrate the idea of merging of datapath operators with an example. Consider the data flow graph G_2 in Figure 1(a).

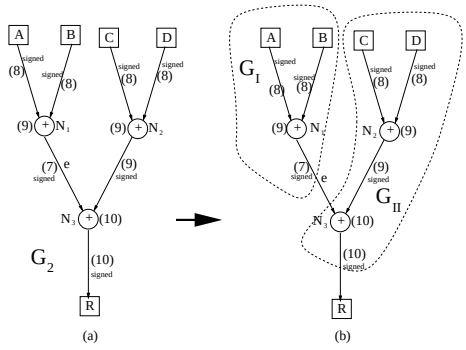

Figure 1: Cluster creation in DFG

Note that output of node N_1 is obtained by truncating the 9 bit sum to 7 bits. Further on edge e, this truncated values gets sign-extended to 9 bits, which is used as an operand for node N_3. Because of extension of a truncated result, the output R of G_2 is not directly expressible as sum of addends *derived* from input signals[2]. Therefore, the whole of G_2 cannot be in the same cluster. Figure 1(b) shows the maximal merging possible in this DFG. Such a situation where a signal is truncated and then subsequently extended in the downstream computation creates a mergeability bottleneck and forces a bounadry for merging.

The following two essential conditions are required for a set of datapath operators in a DFG to be viewed as a cluster :

1. The subgraph formed by the operators is a *connected induced* subgraph with a unique output.
2. The value of the output signal at the unique output node, is definable as a *mergeable* function of inputs to the cluster. For example, for the optimization algorithm of [2] and also for most of discussion in our paper, this function is a sum

[2]Here an addend is said to be derived from an input signal if it is obtained by truncation, extension or 2's complement of the input signal

of products of signals derived from inputs. Note that since a product operation can be implemented as sum of multiple partial products, we can view a sum of products of signals as a sum of addends, where the partial product of inputs form the addends.

4. REQUIRED PRECISION AND RELATED DFG TRANSFORMATION

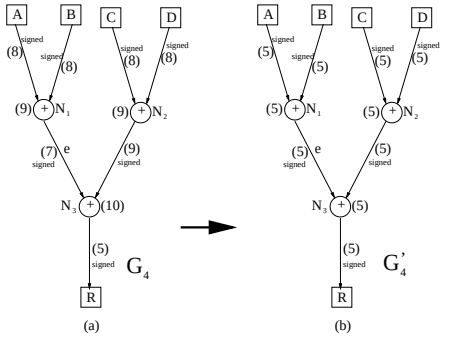

Figure 2: Small required precision implies mergeability

Before formally defining *required precision*, we explain the notion with a simple example. Consider the graph G_4 in Figure 2(a), which is same as G_2 (Figure 1(a)), except for the width 5 of output. Since only 5 least significant bits of final sum need to be generated, therefore for each intermediate result, only 5 least significant bits need to be generated; in other words the required precision of of every signal in the graph is 5 bits; the higher significance bits are superfluous. Hence no extension is required on edge e and G_4 is completely mergeable. In fact, the graph can be transformed to $G_4^{'}$ (Figure 2(b)), which has smaller widths and enhanced mergeability of operators compared to G_4. Note that, though in this example, the width of the output signal is used to update the width of the operators of graph; in general, width of any node or edge inside the data-flow graph can be used to transform the widths of nodes and edges in the fanin cone of the given node (or edge respectively).

In the following, we define required precision for signal at every port in a DFG. The definition is recursive and bottom up i.e. the ports on the output nodes form the base case.

DEFINITION 4.1. **Required precision :** *For an input or output port p, the required precision $r(p)$ for the signal entering or leaving the port respectively, is given by the following rules :*
- *For input port p of an output node N : $r(p) = w(N)$.*
- *For input port p of a non-output node N : $r(p) = min\{r(p_o), w(N)\}$. Here p_o is the output port of N.*
- *For output port p of a node N :*

$r(p) = max_{e \in outedges(N)}\{min\{w(e), r(p_d)\}\}$
Here p_d denotes the input port at the destination node of edge e.

Another way to view required precision is as follows : given a port in a data-flow graph, for every directed path from the port to an output node, find the minimum width of any node or edge on the path; required precision is the maximum of this value over all of these directed paths. Intuitively, if required precision of a signal is n, it means, not more than n least significant bits of the signal are needed to completely define the signals at every output node in the fanout cone of the port. The remaining higher order bits of the signal get truncated by some intermediate operation or explicit truncation on directed every path and hence are superfluous.

As demonstrated by the example given in the beginning of this section, analysis of required precision of a data-flow graph, can potentially reduce the required width of operators and operands and also enhance the mergeability of operators. The following theorem defines the width reduction that can be done based on required precision analysis.

THEOREM 4.2. *Consider a transformation that changes the width of nodes and edges in a DFG such that*

$$w(n) = \min\{w(n), r(p_o)\}$$

and

$$w(e) = \min\{w(e), r(p_d)\}$$

where p_o is the output port of node n and p_d is the destination port of edge e. The functionality of the DFG is preserved by this tranformation.

The transformation indicated in Theorem 4.2 can be efficiently performed by processing the nodes of a DFG in reverse topological order.

5. INFORMATION CONTENT AND RELATED TRANSFORMATION

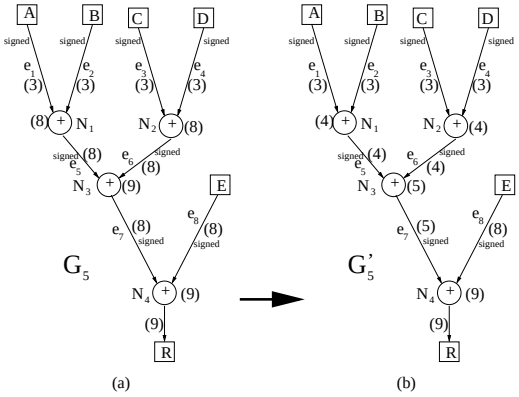

Figure 3: Low information content implies increased mergeability

Once again, we start with a simple example to understand the notion of information content. Consider graph G_5 (Figure 3(a)). Note that edge e_7 seems like a potential boundary of merging, because it is sign-extending an 8 bit truncated sum. However since A, B, C, D have small bitwidths, the 8-bit results of nodes N_1 and N_2 are simply sign extensions of 4 bit sums. Carrying this analysis one level further, result of N_3 is sign-extension of 5 bit sum. This means, the combination of widths of node N_3, edge e_7 and node N_4 does not really imply a sign-extension of a truncated result; in fact, the operand entering N_4 via edge e_5 is a sign extension of 5 bit sum. Note that the preceding analysis allows us to replace G_5 with functionally equivalent graph $G_5^{'}$, which has smaller widths for some nodes and edges (Figure 3(b)). Further it also allows us to conclude that output R is expressible as sum of sign-extensible inputs A, B, C, D, E and the entire graph is mergeable.

This example illustrates that careful analysis of essential *content* of *information* in the result of every operator node, can in some situations, allow to merge operators, which otherwise seemed unmergeable. Also, as noted in the context of preceding example, the same analysis also allows to reduce the widths of datapath operators which are working on operands with small information content.

We now formally define the notion of *information content* and provide an efficient algorithm for computing an upper bound on the information content of signals at every port of a DFG. We also show how information content can be used to prune the widths of nodes and edges in the DFG safely.

DEFINITION 5.1. **Information content :** *Information content of a signal in a DFG is the tuple $\langle i, t\rangle$ of the smallest possible non-negative integer i and an extension type $t \in \{0, 1\}$ (i.e. {unsigned, signed}) such that : for all possible values of inputs to the DFG, the signal is a t-extension of its i many least significant bits. For a port p, we use $\langle i(p), t(p)\rangle$ to denote the information content of the signal entering (or leaving) the port if the port is an input (resp. output) port.*

DEFINITION 5.2. **Intrinsic information content** *of a node is the information content of its result signal in terms of the information content of its operands, assuming the operation at the node is done without any loss of information. For example, intrinsic information content of addition of operands with information contents $\langle m_1, unsigned\rangle$ and $\langle m_2, unsigned\rangle$ is $\langle \max\{m_1, m_2\} + 1, unsigned\rangle$.*

The following theorem shows that the problem of exact computation of first component of information content is hard.

THEOREM 5.3. *The problem of computing the first component of information content of signals in any given DFG with $+$, $-$ and $\times$ operators is NP-hard.*

While computing the exact value (say $\langle i, t\rangle$) of information content is hard; it is still possible to efficiently compute an upper bound on information content i.e. a tuple $\langle i', t'\rangle$, where $i' \geq i$ such that the signal is a t'-extension of its i' many least significant bits.

Notation : Henceforth we use the notation $\hat{i}(p)$ (similarly $\hat{i}(N)$ and $\hat{i}_{int}(N)$) to denote upper bounds on the information content $i(p)$ of a port. We will also use the term information content to mean upper bounds on information content and will often let the notation (i.e. $\hat{i}(p)$ versus i(p)) clarify what we mean in any particular context.

The following lemma gives expression for upper bounds on intrinsic information content of the common datapath operators. These expressions are used later in the algorithm which computes information content for signals in a given DFG.

LEMMA 5.4. *Let $\langle i_1, t_1\rangle$, $\langle i_2, t_2\rangle$ denote upper bounds on information contents of inputs of binary operators of addition(+), subtraction(-), multiplication($\times$) and unary minus($-_u$). Then :*
$\hat{i}_{int}(+) = \langle \max\{i_1, i_2\} + 1, t_1 | t_2\rangle$;
$\hat{i}_{int}(-) = \langle \max\{i_1, i_2\} + 1, signed\rangle$;
$\hat{i}_{int}(\times) = \langle i_1 + i_2, t_1 | t_2\rangle$;
$\hat{i}_{int}(-_u) = \langle i_1 + 1, signed\rangle$.

Since information content of a signal at output edge of an operator node depends on width of operator node and information content of input operands of the operator node, we compute the information content of signals in a given DFG in a top-down order i.e. starting at input nodes and finishing at output nodes. Two key steps in this computation are :
(i) *Propagating information content across an operator node* : Given information content for inputs ports of an operator node; compute information content for the output port of the node.
(ii) *Propagating information content across an edge* : Given information content for the source port of an edge; compute information content for the destination port.

The information content at the output port of a node is the smaller of the intrinsic information content of the node and its width. For propagating information content across an edge, notice that if the sign of the information content and the edge are the same, then the width of the information content across the edge is the smaller of i and w_e. In the scenario where the signedness type t of the information content at source port differs from signedness type $t(e)$, an interesting case arises when $t =$ unsigned and $t(e) =$ signed. In this case, if there is a *strict* extension of the information content across the edge (i.e. $w(N_1) > i$ and $w(e) > i$) then the first component of the information content is i and the signedness is unsigned. Hence, even though the edge is signed, in this case we can view the data going into the destination node as unsigned because it will always have 0s in the most significant bits beyond the i least significant bits.

5.1 Width Pruning Using Information Content

Intuitively, information content captures the minimum number of bits that need to be retained at a particular port *without altering the functionality of the design*. Hence, we can use information content to reduce the widths of nodes and edges in the graph, in the case when widths exceed the information content. We first need the definition of a new type of node i.e. *extension node*, which gets created in DFG, during the transformation.

DEFINITION 5.5. *An extension node is an operator node which performs an extension on the input signal. The node has two attributes : width and signedness (denoted by $w(N)$ and $t(N)$ for node N), such that the result of extension operation is defined as follows :*
(i) if $w(N) > w(e_{in})$ (where e_{in} is the unique input edge of the node), then result is a $w(N)$ bit extension of the signal at the destination port of e and the type of extension is same as $t(N)$.
(ii) if $w(N) \leq w(e_{in})$, then result is the $w(N)$ many least significant bits of the signal on destination port of e.

LEMMA 5.6. *If the intrinsic information content of an operator node N is $\langle i, t\rangle$ and $w(N) > i$, then the following transformation can be done without changing functionality of the DFG : decrease width of N to i; remove all the outedges of N, connect the output port of N to a new extension node and connect the removed outedges of N to the output port of the new extension node; the width and signedness type of the edge connecting N and the new extension node is $\langle w(N), X\rangle$ (where X means either of signed or unsigned is fine); the width and signedness type of the new extension node are $w(N)$ (old value) and t respectively.*

LEMMA 5.7. *If the information content at the destination port of an edge in a DFG is $\langle i, t\rangle$, then the width and sign type of the edge can be changed to i and t without changing the functionality of the DFG.*

The width transformations defined by the above Lemmas can be performed while evaluating the information content in topological order from inputs towards outputs.

5.2 Refining Information Content Upper Bounds

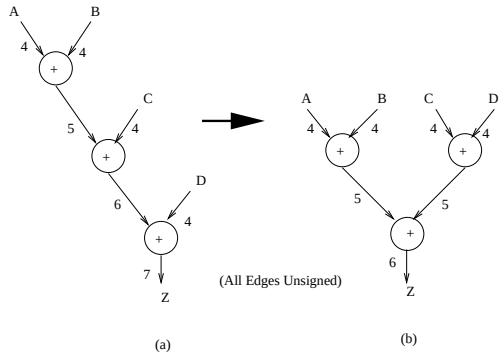

Figure 4: Refining information content upper bounds by safe rebalancing.

There are situations, in which, a safe rebalancing of a subgraph of a DFG, can allow to obtain tighter (i.e. smaller) values of upper bounds on information content of signals; in turn this can allow for potentially greater merging and smaller widths of operators.

For example, consider the DFG shown in Fig. 4(a) which could be part of a bigger DFG. Since the adders form a skewed tree, the algorithm for computing information content computes $\langle 7, 0\rangle$ as the upper bound on information content of signal Z (i.e. output port of last adder). However, if we were to rebalance the tree as shown in Fig. 4(b), the upper bound computed is $\langle 6, 0\rangle$. Note

that this rebalancing of the subgraph in a DFG did not alter its functionality. Therefore once a subgraph has been identified as safely rebalanceable, we can try to recompute upper bounds on the output of the subgraph using a more balanced ordering of operations in the graph. Note that, we need not actually rebalance the nodes and alter the graph; but only need to come up with a more balanced ordering of operators in order to compute tighter upper bounds. In order to identify safely rebalanceable subgraphs, we can use the following observation.

OBSERVATION 5.8. *In a DFG, a cluster obtained from mergeability analysis is a safely rebalanceable subgraph (e.g. G_I, G_{II} in Figure 1(b)). This is because, by its definition, output of cluster is expressible directly as sum of products of input signals.*

OBSERVATION 5.9. *Given a DFG consisting of addition, subtraction, multiplication and unary minus; let a cluster be such that its unique output is expressible as a sum of constant multiples of addends (e.g. $z = 5*b - 4*d + 3*f$), then such a cluster is a safely rebalanceable subgraph. Here we view each constant integer product as multiple addends coming from the same signal (e.g. $5*b$ is $b+b+b+b+b$ and $-4*d$ is $(-d)+(-d)+(-d)+(-d)$); so the output can be viewed as sum of addends derived from input signals.*

The above discussion suggests that after identifying clusters using an initial mergeability analysis, we can recompute the information content of the output of the clusters by rebalancing them. Further, if this recomputation leads to reduction in the value of the width component of information content, it may allow further merging of operators.

We now consider the computational problem of computing tighter upper bounds on information content of a cluster, which represents a sum of constant multiples of inputs. The following algorithm **Huffman_Rebalancing** takes an expression representing sum of constant multiples of input signals and computes an upper bound on integer value of information content of the output signal using the optimal ordering of operations. This is modeled after the minimum redundancy coding algorithm of [1].

Algorithm **Huffman_Rebalancing**
(Input : An expression representing sum of constant multiples of input signals. Upper bounds on information contents of the input signals are known.)
(Output : Upper bound on information content of output signal of the expression.)
Step 1 : Create a priority heap structure H of integers as follows : for each term $c * I$ in the expression (where c is an integer constant and I is an input signal; put c copies of numeric value of information content of I.
Step 2 :

```
while (H has more than one value) {
  min1 = extractMin(H);
  min2 = extractMin(H);
  InsertValue(H, max{min1,min2}+1);
}
return extractMin(H); /* Return the single remaining value in H. */
```

End Algorithm

The following Theorem shows that the above algorithm computes upper bound on information content which is the best possible among all possible orderings of operations..

THEOREM 5.10. *Among all possible orderings of operations in an expression representing sum of constant multiples of inputs; the ordering defined by* **Huffman_Rebalancing** *algorithm gives the tightest possible upper bound on information content of expression result.*

6. ALGORITHM FOR COMPUTING MAXIMAL CLUSTERS

We now return to the problem of partitioning a DFG into clusters. We describe our algorithm for computing maximal clusters based on the analyses of required precision and information content. The algorithm repeatedly does a bottom up traversal (outputs to inputs) of the DFG and identifies *break nodes* i.e. every operator node N such that N is not mergeable with at least one of the operators at the destination of its outedges. This defines a partitioning of the graph into clusters; the clusters are connected components obtained by removing those outedges of every break node, whose destination nodes are not operator nodes. Following conditions are used to identify break nodes.

Conditions for identifying break nodes : Assume that input DFG G has been transformed based on analysis of required precision and information content . Then an operator node N of G, is a break node if one or more of following conditions hold :

1. *Safety Condition 1 :* For some outedge of N, the destination node is an extension node.
2. *Safety Condition 2 :* Let $p_1, \ldots, p_m$ be the destination ports of outedges of N. Let $r(p_i)$ denote the required precision of signal for each p_i. Then $\min\{i_{int}(N), \max\{r(p_1), \ldots, r(p_m)\}\} \leq w(N)$.
3. *Synthesizability Condition 1 :* For some outedge of N, the destination node has multiplication operator.
4. *Synthesizability Condition 2 :* There is a node N' such that every directed path starting at N goes through N' and there are no break nodes between N and N' on any of these paths.

Synthesizability condition (2) ensures that every cluster has a unique operator node providing outputs; synthesizability condition (1) ensures that this unique output is expressible as sum of products of inputs to the cluster. Then each cluster can be synthesized as a sum of addends.

Before describing the clustering algorithm, we need to clarify one more technical issue. If the algorithm for information content computation encounters an extension node, created by the previous iteration of information content computation; it needs to propagate information content across the extension node. The following observation describes how this can be done.

OBSERVATION 6.1. *Let N be an extension node and let $\langle i, t\rangle$ be upper bound on information content at its input port. Let e be the inedge of N. Then an upper bound $\langle i_o, t_o\rangle$ on the output port of N can be defined as follows :*

(i) if ((t == t(N)) OR ((t == unsigned) AND (t(N) == signed))) then
$i_o = \min\{i, w(N)\}$; $t_o = t(N)$;

(ii) if ((t == signed) AND (t(N) == unsigned)) then
$i_o = \min\{w(e), w(N)\}$; $t_o = t(N)$;

Note that after initial computation of required precision and information content , the algorithm for maximal merging enters the iterative mode. Every iteration defines a partitioning based on current values of information content and uses current set of clusters to compute tighter upper bounds on the information content of the output signals of clusters. Whenever the value of information content at output of any cluster change, another iteration of cluster definition is done with the anticipation that smaller information content could lead to more mergeability and result in bigger and fewer clusters. This way the algorithm converges to a partitioning with maximal safe clusters.

7. EXPERIMENTAL RESULTS

The new DFG partitioning algorithm has been implemented and tested, as a dfg optimization and datapath operator merging step in the BuildGates synthesis tool of Cadence Design Systems. We used datapath intensive RTL testcases and collected experimental data on the performance of the algorithm and compared

with results obtained using an older implementation of cdfg partitioning algorithm. The older algorithm did mergeability analysis using criteria similar to "leakage of bits" notion of [2] and without doing any transformations based on information content and required precision.

Using TSMC 0.25 micron technology cell library, we collected two types of performance data :
(i) Longest path delay and area of the netlists obtained after synthesis but before any timing driven gate level logic optimization.
(ii) Runtime of timing driven gate level logic optimization done on netlists obtained from synthesis.

In Tables 1 and 2 respectively, we present the above two types of data from five datapath-only testcases. To highlight the impact of operator merging in datapath synthesis, Table 1 also includes the data obtained using a synthesis flow which does not do any operator merging. When the non operator-merging based flow was used, the runtimes of logic optimization were much larger than those with operator-merging based flows; so for fair comparison, we did not include their runtime in Table 2. To further compare of the quality of the final netlists generated using old and new merging algorithm, we have also included in Table 2, the data on final longest path delay and final area after timing driven logic optimation. All delay numbers are in nanseconds and the area numbers are scaled down by a factor of 100.

Note that to collect data for both tables, we set the arrival times at all inputs in each testcase to 0.

Table 1

Testcases →		D_1	D_2	D_3	D_4	D_5
Del. (ns)	No mg	14.47	18.01	33.59	29.23	25.89
	Old mg	13.04	11.97	29.90	28.13	25.89
	New mg	12.73	11.07	29.27	16.97	15.57
	% red.	2.38	7.52	2.11	39.67	39.86
Area (unit)	No mg	93.8	79.3	1866	490	279
	Old mg	91.7	66.6	501	397	225
	New mg	90.3	66.6	476	43	33.3
	% red.	1.53	0	5	89.2	85.2

Table 2

Testcases →		D_1	D_2	D_3	D_4	D_5
Target delay (ns)→		5.0	4.0	21.0	10.5	14.0
Opt time (sec)	Old mg	470	1031	26	118	21
	New mg	6.8	208	17	2.2	1.3
	% red.	98.5	79.8	34.6	98.1	93.8
End Del.	Old mg	4.99	4.35	20.7	10.5	13.9
	New mg	4.99	3.98	20.9	9.1	12.2
End Area	Old mg	161	155	377	609	259
	New mg	142	118	363	44	35

Testcase D_1 and D_2 were created using multiple addition operations which are potentially mergeable. These addition operations did not have any redundant widths in RTL code; so the first pass of information-analysis leads to clusters which are not distinguishable from those created by the old merging algorithm. However, the post-clustering information analysis based on optimal reordering of operations, which is done by the second or subsequent iteration of the new merging algorithm, allows to infer smaller information content for output signals of clusters. This allows the second or subsequent iterations to merge the set of clusters created in previous iteration into bigger and fewer clusters. This reduction in number of clusters, leads to the better longest path delay and area values after initial synthesis. Since there were no apparent redundant widths in RTL, the gains seen after the initial synthesis do not seem as large as D_4 and D_5. Nevertheless during timing driven logic optimization, we see considerable advantages of creating larger clusters, and see significantly smaller runtimes.

Testcases D_4 and D_5 were created with lot of redundancy in the bit widths of intermediate wires in RTL, to test the effect of information-analysis based width reduction on timing and area of netlists. In these testcases, the new merging algorithm was able to prune the redundant widths to the minimum required, and this in turn helped in reducing the number of clusters created. As a result, we note that the reduction in longest path delay and area after the initial synthesis is quite significant. This also translates to drastic reduction in the runtime of the timing driven logic optimization for these two testcases, as seen in Table 2.

Testcase D_3 represented a sum of products of sum computation, where information-based-analysis allowed the new merging algorithm to prune with widths of outputs of products and merge them with the the final addition.

The above results demostrate the benefits of using analyses of required precision and information content of signals in DFGs for operator merging based datapath synthesis.

8. CONCLUSION

We have presented new algorithmic techniques for analyzing and optimizing DFGs consisting of datapath operators with the objective of increasing the scope of operator merging and reducing the area and delay of the netlists obtained from synthesis. Our experimental results show that use of these techniques can reduce the area and delay of the netlists and considerably reduce the effort required by timing optimization to meet timing constraints for datapath intensive designs.

9. REFERENCES

[1] D. A. Huffman, A method for the construction of minimum-redundancy codes, *Proceedings of the IRE*, 40(9), 1952, pp. 1098–1101.

[2] T. Kim, W. Jao, S. Tjiang, "Arithmetic Optimization using Carry-Save-Adders", *Proceedings of the 35th Design Automation Conference*, 1998, pp.433-438.

[3] A.R. Omondi, "Computer Arithmetic Systems : Algorithms, Architectures and Implementations", *Prentice Hall International Series in Computer Science*, 1998.

[4] J. Um, T. Kim, C.L. Liu, "Optimal Allocation of Carry-Save-Adders in Arithmetic Optimization" *Proceedings of International Conference on Computer Aided Design*, 1999, pp.410-413.

[5] J. Um, T. Kim, C.L. Liu, "A Fine-Grained Arithmetic Optimization Technique for High-Performance/Low-Power Data Path Synthesis" *Proceedings of the 37th Design Automation Conference*, 2000, pp.98-103.

[6] C. S. Wallace, "A suggestion for a fast multiplier" *IEEE Trans. Electron. Comput.*, Feb. 1964, vol EC-13, pp.14-17.

[7] N. Weste, K. Eshraghian, "Principles of CMOS VLSI Design - A System Perspective" *Addition Wesley Publishers*, 1985.

[8] M. Willems, V. Bursgens, H. Keding, T. Grotker, H. Meyr, "System Level Fixed-Point Design Based on an Interpolative Approach", *Proceeding of the 34th Design Automation Conference*, 1997, pp. 293–298.