An Automated flow for Arithmetic Component Generation in Field-Programmable Gate Arrays

ALASTAIR M. SMITH, GEORGE A. CONSTANTINIDES, PETER Y. K. CHEUNG
Imperial College London, UK

State of the art configurable logic platforms, such as Field-Programmable Gate Arrays (FPGAs), consist of a heterogeneous mixture of different component types. Compared to traditional homogeneous configurable platforms, heterogeneity provides speed and density advantages. This is due to the replacement of inefficient programmable logic and routing with specialised logic and fixed interconnect in components such as memories, embedded processor units and fused arithmetic units. Given the increasing complexity of these components, this paper introduces a method to automatically propose and explore the benefits of different types of fused arithmetic units. The methods are based on common subgraph extraction techniques, meaning that it is possible to explore different subcircuits that occur frequently across a set of benchmarks. A quantitative analysis is performed of the various fused arithmetic circuits identified by our tool, which are then automatically synthesised to an ASIC process, providing a study of the speed and area benefits of the components. The results of this study provide bounds on the performance of heterogeneous FPGAs: by incorporating coarse-grain components which match the specific needs of a set of benchmarks we show that significant improvements in circuit speed and area can be made.

Categories and Subject Descriptors: [**Hardware Engineering**]: Reconfigurable Computing—*documentation*

General Terms: FPGA, common subgraph, reconfigurable logic

1. INTRODUCTION

For embedded systems, reconfigurable devices provide designers with high throughput and cost effective platforms. When designing reconfigurable devices, the architects must be aware of the types of circuit that users intend to map onto the platform. This means that device architectures must be designed with the performance of the different systems for which they are intended in mind. Given the widespread use of reconfigurable architectures for high throughput computation, there have been several advances in reconfigurable chip design for this domain, which is particularly evident in Field-Programmable Gate Arrays (FPGAs).

Modern FPGAs consist of a variety of different resource types to implement a range of functionalities. Logic resources based on Lookup tables (LUTs) are used to implement fine grain operations, and can be configured to implement virtually any digital circuit. However, for some functions, the fine grain fabric in FPGAs is inefficient for several reasons. The lookup tables can be inefficient when compared to gates for particular logic functions. More critically, the routing fabric required to connect blocks uses programmable switches, which are slow and consume a considerable amount of area when compared to fixed connections. To speed up and improve the logic density of particular types of computation, embedded memories and fused arithmetic blocks have been incorporated into FPGA fabrics [Altera Corporation 2005b], [Xilinx Inc. 2004a].

This paper focuses on fused arithmetic components in FPGAs. An example of a

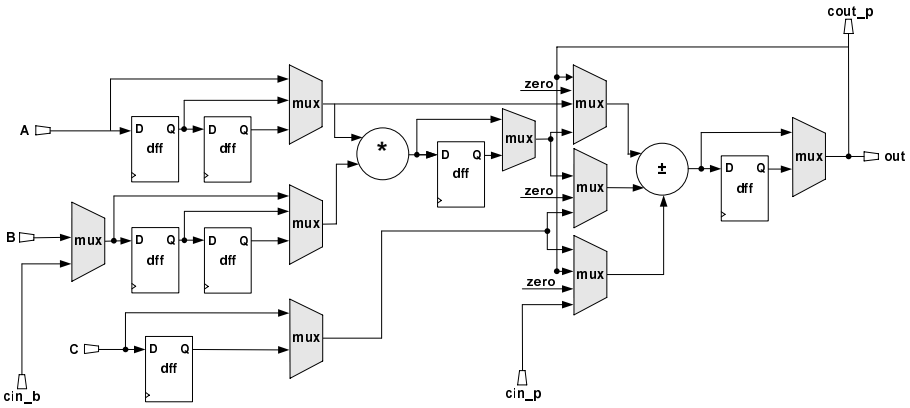


Fig. 1. A simplified view of the Xilinx Xtreme DSP Slice. The MUX select lines are driven by configuration bits.

commercial fused arithmetic unit is the Xtreme Slice [Xilinx Inc. 2004b], evident in Xilinx's Virtex 4 family of FPGAs. A simplified structure of the Xtreme Slice is shown in Figure 1. The configuration multiplexers are one of the central features of this embedded core, providing flexibility that allows support for a wide variety of arithmetic functions. However, flexibility has an associated cost with regards to the metrics of delay, area and power consumption. One of the main aims of this paper is to quantify this cost.

This paper proposes a method for automatically discovering potential fused arithmetic components by examining common connection patterns between circuit netlists. In doing so, it is possible to identify complex computation patterns leading to specialised embedded components that are more efficient than existing FPGA resources. A tool has been developed to create silicon cores from representations of the common connection patterns. These cores can be directly compared with those currently on FPGAs, thus indicating potential speed and density advantages that could improve future FPGA architectures.

The main contributions of the paper can be summarised as follows:

—A methodology for extracting arithmetic subcircuits that occur frequently across a set of benchmarks [Smith et al. 2007].

—Introduction of a logic optimisation phase, to increase the discovery of common subgraphs.

—A study of common arithmetic subcircuits that occur across a variety of benchmark circuits.

—A quantitative analysis of the speed and area tradeoffs of common arithmetic circuits, including comparisons to a commercial 90nm FPGA.

The remainder of this paper is organised as follows. In Section 2 related work is discussed. The problem of common subgraph extraction for fused arithmetic component generation is formally defined in Section 3. Section 4 details the design flow and algorithm used to extract commonly occurring arithmetic subcircuits.

Section 4.3 presents the methodology that is employed to synthesise and compare the commonly occurring patterns. Comparisons between subcircuits implemented in the generated silicon cores and those implemented in a commercially available device are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. BACKGROUND

Considerable research effort has been focused on configurable architecture exploration in recent years. It has been estimated that designs implemented on heterogeneous FPGA architectures are on average approximately 20 times larger and 3-4 times slower than when implemented as an ASIC [Kuon and Rose 2007]. In order to close this gap, there are various aspects of configurable architectures that can be targeted: routing fabric, fine grain logic fabric and coarse grain fabric. In this paper the focus is on improving the coarse grain arithmetic blocks, with a potential knock-on benefit in the reduction of routing requirements. It is the intention of this paper to examine how parts of a design can be improved by automatically detecting common arithmetic patterns in benchmarks and implementing them as their own hard block in silicon.

Detecting common connection patterns is a well known problem, and has been employed in fields such as molecular chemistry, ASIC template generation [Chowdary et al. 1999] and custom instruction set generation [Cone et al. 1977; Cong et al. 2004; Kastner et al. 2002]. In the context of the paper we present here, existing work involving custom instruction set generation is particularly relevant, as it deals with arithmetic operations commonly implemented by field programmable hardware.

In [Cong et al. 2004], subgraph extraction is used in instruction set generation for Application Specific Instruction set Processors (ASIP) implemented on reconfigurable fabrics. The method employed for the problem is hierarchical. First C codes are transformed into control dataflow graphs (CDFGs) and a pattern library is generated by examining each application CDFG. This pattern library is then examined to ascertain the potential speedup of the new custom instructions (patterns), subject to a set of constraints (area and I/O).

Similarly, in [Kastner et al. 2002], subgraph extraction is used in custom instruction set generation for soft processors implemented in reconfigurable fabrics. The method uses template generation using a single benchmark to create a library of subgraphs. The algorithm uses a profiling step to examine potential subgraphs and locally optimal graph covering methods in application mapping.

ISEGEN [Biswas et al. 2005] also uses subgraph extraction for custom instruction set generation. The methods employed use min-cut partitioning heuristics in order to minimise the external communication to the custom instruction units proposed by the framework. Contrary to our proposed framework, [Biswas et al. 2005] focusses on multiple rather than single output graphs. Our discussion of this is elaborated upon in Section 3.

There have also been efforts to use common subgraph detection methods for the purpose of discovering hard cores for FPGAs [Aravind and Sudarsanam 2005], which presents a methodology from the point of view of reducing area and configuration overheads. However, in this paper we present, for the first time, a quantitative anal-

ysis of possible components identified by common subgraph extraction, including a comparison of their speeds and areas in both FPGA and ASIC for 90nm technology.

Configurable architecture generation has also been approached in the context of coarse grain configurable devices. For instance in [Compton and Hauck 2007] architectures are generated for a specific set of benchmarks, and methods are employed to reduce the number of multiplexers and connecting wires. Similar to [Compton and Hauck 2007], we focus on low-level hardware optimisations, but concentrate on generating individual components that can be inserted into reconfigurable fabrics. Moreover, we base our methods on formal techniques and focus on commercial FPGA architectures, consisting of a mixture of coarse and fine grain components.

The problem of common subgraph discovery encompasses that of technology mapping. Traditional technology mapping algorithms for homogeneous fine grain FPGAs, consisting of lookup tables, considers how to pack logic netlists into appropriately sized functional blocks. This is a well researched topic, exemplified by FlowMap [Cong and Ding 1994] and SIS [Sentovich et al. 1992]. With advances in FPGA technology, synthesis must also consider heterogeneous components such as embedded memories and complex DSP blocks, capable of many different fused arithmetic operations.

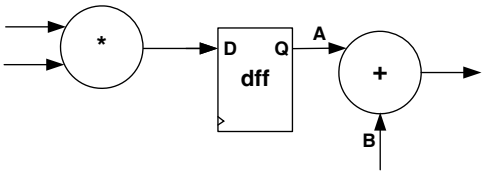
The problem of technology mapping to heterogeneous FPGAs is particularly problematic, as it is difficult to infer components such as DSP blocks and multipliers from logic netlists. Thus, recent efforts have attempted to infer such components from HDL descriptions. Odin [Jamieson and Rose 2005] is an example of a tool for technology mapping to modern heterogeneous FPGAs from Verilog code, and is used in this paper to parse benchmark circuits into their constituent components. A graph matching algorithm, similar to that employed by Odin, is also used to determine which benchmark structures can then be matched by the common subgraphs.

3. PROBLEM DEFINITION

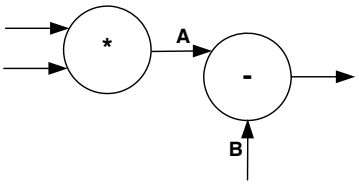
In this paper, we focus on the common subgraph extraction problem in the context of arithmetic circuits. This motivates the following definitions and problem formulation.

Definition 1. A *word-level netlist* (*netlist* for brevity) is a labelled graph $G = (V, E, T)$. V is the (unordered) *node set*, $E \subseteq V \times V$ is the (ordered) *edge set*, and $T \subseteq V \times \mathcal{T}$ is a *type function*.

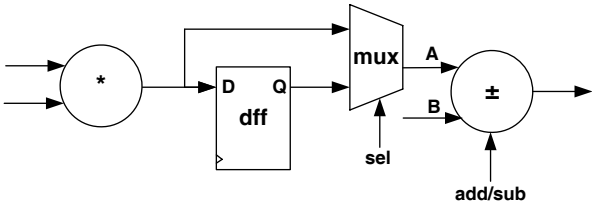
We use word-level netlists as formal representations of benchmark circuits after parsing and elaboration of Verilog, where the node set corresponds to word-level arithmetic operations such as addition (a *type*). The edge set corresponds to the flow of data between these operations. An order is associated with the edge set in order to express precedence and non-commutative operations, for example the netlist computing $(a+b)-c$ is not the same as the netlist computing $c-(a+b)$. The type set $\mathcal{T}$ can be considered as the set of all atomic computational nodes available in the elaborated Verilog, such as $\mathcal{T} = \{add, mult, reg, sub\}$, and the type function T associates nodes with types. We further identify a subset $\mathcal{T}^{NC}$ of types for which the ordering of inputs matters, *e.g.* non commutative arithmetic operations such as subtraction.



(a) Part of a benchmark DFG.



(b) Part of an alternative benchmark DFG.



(c) An example of how (a) and (b) can be combined into a single common subgraph.

Fig. 2. Flexibility detection in Common Subgraph Extraction.

Definition 2. A *subgraph* $G' = (V', E', T')$ of a netlist $G = (V, E, T)$ is a graph with $V' \subseteq V$, $E' \subseteq E$, and $T' \subseteq T$, such that
$\forall u \forall v \left[(u, v) \in E \wedge u \in V' \wedge v \in V' \Leftrightarrow (u, v) \in E'\right]$, and such that the order on E is preserved in E' for edges from nodes with types in $\mathcal{T}^{NC}$.

This definition captures the concept of 'part of a computation', while the final conditions ensure that (i) if two nodes are part of a subgraph, then so are the relevant edges, and (ii) if the order of inputs to a computation is important, then it is preserved.

Definition 3. Two graphs are said to be *isomorphic* iff they are identical up to a relabelling of the vertex set.

Isomorphism is useful in this context, because if two word-level netlists are isomorphic, they compute the same function.

Definition 4. Given two netlists G_1 and G_2, a *common subgraph* G' is a netlist that is isomorphic to a subgraph of G_1 and also isomorphic to a subgraph of G_2.

A common subgraph is therefore a potential candidate for implementation as a hard embedded core in a reconfigurable fabric, since it represents functionality shared across two or more benchmark netlists. However, we often aim to be more flexible in the extraction of embedded cores; for example Figures 2(a) and (b) show

two subgraphs exhibiting multiply accumulate functionality. Firstly, we may wish to combine the add and subtract functionality into a combined adder/subtracter, which makes sense from a hardware point of view. After this merging procedure, if two operations differ only in their latency, we may wish to add optional pipeline registers, allowing both operations to be mapped to the same core Figure 2(c). This observation motivates the following definition.

Definition 5. A *contraction* $G' = (V', E', T')$ of a graph $G = (V, E, T)$ with respect to a *contraction set* $C \subseteq \mathcal{T}$ is a graph with $V' \subset V$ and for which (i) there is a one-to-one correspondence between edges in E' and paths $(v_1, v_2, \ldots, v_n)$ in G for which $T(v_i) \in C$ for $2 \leq i \leq (n-1)$, (ii) E' inherits the order from the final edges (v_{n-1}, v_n) in E, (iii) $T' \subseteq T$.

A contraction is thus a way of absorbing a number of nodes of a path into a single edge, allowing the common subgraph algorithm to bypass certain information that is necessary for synthesis of the graphs only. We therefore have a very flexible conceptual framework for defining a candidate hard embedded core: it is any common subgraph of the respective minimal contractions of two word-level netlists. By setting $C = \{reg\}$ we can, for example, achieve optional pipeline registers. For the remainder of this paper, this is the contraction set used.

A further constraint on the common subgraph extraction algorithm is that the common subgraph should have only one output, a problem known as multiple input single output (MISO) graph extraction. In configurable hardware design there are several considerations that are important in hardware generation. Configurable routing wires are capacitance heavy, and logic resources require relatively large multiplexers to connect their inputs and outputs through the FPGA routing fabric. The single-output simplification thus suits this particular context well.

4. COMMON SUBGRAPH EXTRACTION ALGORITHM AND SYNTHESIS

An overview of the tool flow used in this work is shown in Figure 3. The netlists for use in the common subgraph extraction framework are required to be word-level. A combination of the publicly available tools Icarus Verilog [Williams 1999] and Odin [Jamieson and Rose 2005] are used to parse Verilog benchmarks. All technology specific optimisations available in Odin are turned off, as simple arithmetic graphs are required to explore potential embedded arithmetic units. The output of Icarus/Odin is a flattened netlist of the basic components: standard logic operators, such as AND/OR; multiplexers; registers; arithmetic components; memories; and relational operators, *e.g.* greater than/less than.

The netlists produced by Icarus/Odin are fed through some simple optimisation procedures (discussed in Section 4.1). This allows the discovery of many more arithmetic common subgraphs, by reducing logic between arithmetic components. The resulting netlists are then fed into the common subgraph extraction algorithm (discussed in Section 4.2).

The common subgraph extraction algorithm operates on pairwise combinations of benchmarks. The common subgraphs must thus be examined across the benchmark set to assess the wordlength requirements of incorporating such a block into an FPGA fabric and the benefits of using such a block. Once the wordlength requirements are known, the block can be synthesised.

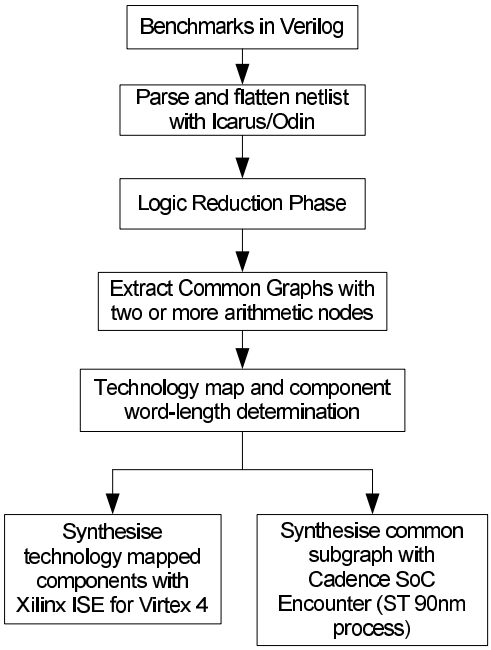
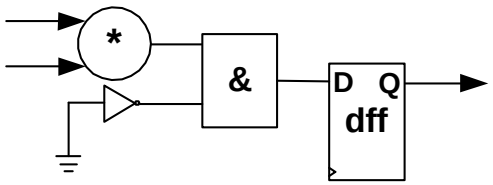


Fig. 3. The design flow.

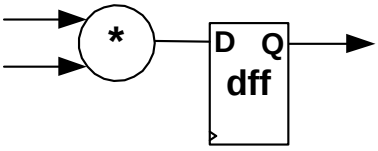
4.1 Constant Propagation

The netlists produced by Odin are optimised for use with heterogeneous architectures, and are intended for use in conjunction with further synthesis and optimisation tools. Thus it is important to perform some logic optimisation on the netlists. The aim of this phase is to remove logic chains between arithmetic components and facilitate discovery of common fused arithmetic subgraphs. This motivates the introduction of one of the new features of our tool over [Smith et al. 2007]: a constant propagation and simplification phase.

The logic optimisation phase introduced is relatively straightforward, and was motivated through observation of the Icarus/Odin netlists. A particularly inefficient part of the verilog parser toolset was found to register input signals. Many of the input signals generated that feed into registers involve propagation of constants through logic chains, a simplified example of such a combination is shown in Figure 4, which clearly can be reduced by applying simple logic identities. Odin netlists involve one and two input gates only, and the identities supported are simple transformations based on constant propagation through standard gates (e.g. $\overline{1.\mathrm{A}} = \overline{\mathrm{A}}$), and logic identities such as $\overline{\mathrm{A}.\mathrm{A}} = \overline{\mathrm{A}}$ The logic identities are applied iteratively until no more gates can be eliminated. The impact of these reductions is discussed in the results section.



(a) An example of a commonly produced structure.



(b) Simplified version.

Fig. 4. A simple example of commonly produced logic types from Icarus/Odin and the simplification of these netlists.

Algorithm 1 : Common subgraph extraction top-level algorithm

Input: G_1, G_2

```
1: G' = ∅
2: for all v_i in G_1 do
3:   for all v_j in G_2 do
4:     if ISMATCH(v_i, v_j) then
5:       add nodes to G'
6:       FAST_MATCH(G_1, G_2, G')
7:     end if
8:   end for
9: end for
```

4.2 Common Subgraph Extraction Algorithm

The common subgraph extraction algorithm used for this work is based on one in [Bunke et al. 2002], and has been developed to operate on large, directed graphs, containing arithmetic components. The algorithm identifies potential fused arithmetic units by finding common graphs between two benchmarks, and a matching algorithm is employed to calculate the frequency of common graphs across the entire benchmark set.

The algorithm for extracting common subgraphs focuses on arithmetic components and is described in Algorithm 1. The algorithm starts by traversing both graphs and finds two matching nodes using the 'IS_MATCH' function, which in this case only has to check that the types of the two nodes (multipliers and adders or subtracters) are ammenable. A new node is then created for the common subgraph structure, which acts as a seed node to grow the rest of the common subgraph. The algorithm then recursively tries to add nodes to this graph.

Algorithm 2, 'FAST_MATCH' describes the recursive part of the algorithm, which adds nodes to the common subgraph. For each node in the common sub-

graph, a node-pair $v' = (v_1, v_2)$ is created and functions $r_1(v')$ and $r_2(v')$ are created to return the appropriate benchmark node from the common subgraph node. The algorithm proceeds by examining each node in the common subgraph and the benchmark nodes to which it refers by employing the mapping functions $r_1(v')$ and $r_2(v')$ for each node in the common subgraph. The procedure must thus determine whether a new node from some combination of the inputs to the benchmark nodes can satisfy the problem constraints. The four problem constraints are as follows: the two potential benchmark nodes matched must be of the correct type; the ordering of nodes must be consistent (*i.e.* if there is a subtraction involved the nodes must be connected on the correct port); that they must both have the same connection pattern within the common subgraph; and that only one node can have an output. The recursive procedure enumerates all possibilities of inputs for each node within the common subgraph to determine whether a new node can be added to the common subgraph. If a node can be created, it is added to the common subgraph along with the appropriate connections and the algorithm continues by recursively trying to add nodes to the common subgraph. On exiting a recursive step, the common subgraph node is removed from the common subgraph so that a different combination of benchmark nodes can be tried in creating a common subgraph node.

In Algorithms 1 and 2, the 'ISMATCH' function is used to test the compatibility of two benchmark nodes to the common subgraph. It must first test the two node types of the pair are compatible. It then has to ensure that adding such a node will satisfy other problem constraints, such as that the outputs of this potential node are not used outside the common subgraph, as the common subgraph is restricted to single-output graphs. The function also has to determine whether by adding this component, and any combination of components connected to and from this component, a graph satisfying the problem constraints can be extracted. In the common subgraph extraction add and subtract nodes are allowed to be matched to create a component with a configuration port.

The algorithm keeps copies of all graphs that satisfy the constraints, and thus terminates when all combinations of seed nodes from the two benchmarks have been examined. The pairwise nature of Algorithm 1 results in $N_{t1}N_{t2}$ comparison operations, where N_{t1} and N_{t2} represent the total number of nodes in the benchmark circuit. The complexity of the recursion (Algorithm 2), is dependent on the size of the largest arithmetic common subgraph N_c. As the size of the common subgraph is expanded a node at a time, the number of comparisons in each branch of the recursion increases linearly, as there is an additional node to compare each of its inputs for. For a common subgraph of size n, the number of computations performed by the recursion f_n is thus $f_n = 2n + k_{n-1}f_{n-1}$, where k_{n-1} represents the number of possible recursive branches in the preceding stage leading to the common subgraph of size n. k_n in the worst case will grow linearly due to the number of free inputs to the nodes in the common subgraph being dependent on the size of the subgraph. However, as the size of the common subgraph approaches N_c, the 'ISMATCH' function prunes the design space minimizing the number of recursive steps. In the worst case the algorithm has exponential time complexity with respect to the maximum common subgraph size. However, in practice the

Algorithm 2 FAST_MATCH: Recursive algorithm to add nodes to a common subgraph

Input: G_1, G_2, G'

```
1: for all nodes u' in G' do
2:   for all input ports i on node r1(u') do
3:     u1 = node on port i of r1(u')
4:     for all input ports j on node r2(u') do
5:       u2 = node on port j of r2(u')
6:       if ISMATCH(u1, u2) then
7:         add node v' = (u1, u2) to G' on appropriate port of u'
8:         FAST_MATCH(G1, G2, G')                    // recursive call
9:         remove v' and associated connections from G'
10:      end if
11:    end for
12:  end for
13: end for
```

size of the common subgraphs is low, resulting in manageable runtimes. Moreover, the number of arithmetic nodes is relatively small, meaning 'ISMATCH' function prunes the design space further. The common subgraphs from all combinations of the benchmark set (17 benchmarks) can be extracted in less than half an hour on a conventional workstation (Pentium 4 2.4GHz, 1 GB memory, running windows XP).

An important consideration for the device architect is the frequency of the common subgraphs. The more frequent the graphs are, the better they are as candidates to be used in the configurable fabric. In this paper, a graph covering approach is taken to this problem. A complete enumeration of potential non-overlapped coverings is made for each benchmark, with the mapping that covers the most nodes being chosen as the maximum covering.

During the process of evaluating the frequency of the common subgraphs it is also possible to obtain information about the required wordlengths of the structures for the process of synthesising the graph as an embedded core. For each benchmark the maximum graph covering is used and the largest wordlength for each component is selected so that the synthesised core is capable of supporting the wordlength requirements of all graph coverings in the benchmark set. Modern synthesis tools are capable of combining multiplier blocks together in order to account for wordlengths beyond that supported by an individual block. This is advantageous for smaller wordlengths, particularly for blocks such as Altera's DSP block [Altera Corporation 2005a], which is fracturable into many combinations of wordlengths other than the maximum. However, for benchmarks with greater precision requirements or dynamic range, blocks with greater wordlengths are particularly advantageous, as they negate the need to employ the slower fine grain logic of an FPGA. This tradeoff will be examined in Section 5.

4.3 Common Subgraph Synthesis

After common subgraph extraction and technology mapping, the tool can be used to compare components implemented in FPGA and ASIC, to ascertain the speed and area tradeoffs. This is done automatically by creating HDL files for each of the common subgraphs and each of the respective technology mapped subcircuits of the benchmarks. These HDL files are then compiled to FPGA and ASIC.

In order to perform synthesis, two design flows have been used. The FPGA design flow used is that for the 90nm Xilinx Virtex 4 FPGA, using Xilinx Integrated Software Environment (ISE) 8.2 [Xilinx Inc. 2006]. The fastest speed grade Virtex 4 LX45 was used. Two runs of synthesis are performed: once to ascertain the number of components required by the structure from the benchmark, and a second time to ascertain the speed. To evaluate speed, the benchmark subcircuit is placed between registers and the critical path delay is observed. These registers ensure that delays associated with IO pads and the required routing do not increase the critical path. The delay from an input register to a logic input is also accounted for to ensure the delay calculated is not skewed by routing delays to and from the FPGA component.

To synthesise the common subgraphs as embedded cores, the Cadence digital IC design suite (version 5.2) was used. This uses a combination of RTL compiler [Cadence Design Systems Inc. 2005b] and SoC Encounter [Cadence Design Systems Inc. 2005a], and incorporates full place and route. The Xilinx Virtex 4 is manufactured in UMC's 90nm technology, but due to the unavailability of UMC's technology files, the ASIC STMicroelectronics 90nm library [STMicroelectronics 2006] was used at the appropriate voltage (1.2V).

The Virtex 4 DSP blocks have been designed at a specific tradeoff point in the area/time space, selected by the manufacturer. Hence to perform a reasonable comparison to the commercial architecture three versions of each subgraph identified by our tool have been synthesised covering a range of potential cores in the speed-area design space. This is done by first optimising for speed: an attempt to force the component to operate at 2GHz is made, which is an unreasonable constraint, but provides us with the fastest core possible given the synthesis flow. The estimated operating frequency of this implementation is ascertained through the synthesis tools. In order to find a balanced area-speed implementation of each core, the constraint on the operating frequency is relaxed to a half of its corresponding fastest core. The minimum area point in the design space is then found by relaxing the timing constraint to a third of the actual operating frequency of the fastest core. These frequencies were found experimentally to evenly cover the area-speed design space for the embedded cores.

5. RESULTS

5.1 Benchmarks and Logic Minimisation

To explore the common subgraph extraction and synthesis methodology, a set of industrial and academic designs were used. These are a subset of those used in [Jamieson and Rose 2005], spanning a variety of application domains such as DSP and scientific computing. Only a subset of the designs are used, as several of the benchmarks in the original suite are essentially replicated with differences only in the memory components. Statistics of each benchmark presented in Ta-

Table I. Summary of the statistics of each benchmark circuit, including the number of arithmetic components that exist, the resource usage when the entire circuit is synthesised to Virtex 4 and the self-similarity of the arithmetic components in the benchmark circuits. Figures in brackets indicate results when no logic minimisation phase was used.

B'mark	Add/Subs		Multipliers		Largest self-similar graph in # of nodes (no logic red'n)	Virtex 4 resource usage	
	#	wordlength min-mean-max	#	wordlength min-mean-max		slice	DSP
boundtop	47	2-7.9-18	0	X-X-X	1 (1)	1017	0
cordic	49	2-7.7-8	0	X-X-X	1 (1)	454	0
diffeq1	4	32-32-32	5	3-25.6-34	7 (7)	191	13
diffeq2	4	32-32-32	5	3-26-32	7 (7)	119	13
fir_scu_rtl	16	19-19-19	17	7-7.5-8	1 (1)	255	2
fir_3_8	3	17-17.3-18	4	8-8-8	7 (1)	34	4
fir_24_16	24	33-33.9-37	25	16-16-16	48 (1)	642	25
frame_buff	17	7-7.8-9	0	X-X-X	3 (3)	437	0
iir	4	3-16.25-29	2	15-15-15	5 (5)	261	5
iir1	16	7-11.6-14	5	6-6.8-10	4 (4)	127	5
oc_54_cpu	26	2-28.6-40	1	17-17-17	2 (2)	1319	1
MAC1	4	4-16.8-29	0	X-X-X	2 (2)	1324	0
MAC2	5	2-28.7-36	0	X-X-X	2 (2)	5138	0
ray_gen	47	2-15.0-18	18	8-11.8-16	14 (7)	1113	18
rt_top	80	18-62.6-80	24	16-24.3-46	7 (7)	9519	54
rs_dec1	6	3-4.2-6	13	5-5-5	2 (2)	446	8
rs_dec2	6	2-5.8-10	9	9-9-9	2 (2)	980	9
Total	358	2-25.9-80	128	2-14.4-46	N/A	22059	155

ble I, including synthesis of the verilog code onto a Virtex 4 device. Some of the benchmarks map particularly well to the Virtex 4 architecture, for example the DSP benchmarks 'iir', 'iir1', 'fir_3_8_8' and 'fir_24_16_16' all take full advantage of the DSP blocks. However, 'fir_scu_rtl' does not, implementing only 8 multipliers in DSP blocks (note that the tools take advantage of the fracturable nature of the DSP blocks in this case).

The common subgraph extraction algorithm works on pairwise combinations of benchmarks. In order to find an upper bound on the size of the subgraphs that exist in the benchmark set, the common subgraph extraction technique was applied to each benchmark with the same benchmark used as both input graphs to the algorithm. This is equivalent to finding the largest arithmetic subgraph for each benchmark. The results of this study are shown alongside the benchmark statistics in Table I. The results show that there is potential for some relatively large subgraphs when compared to the DSP blocks currently used in state-of-the-art FPGAs.

The results in Table I also show the importance of the logic minimisation phase. This is particularly the case with the FIR filter benchmarks: inefficiently generated combinatorial logic exists between arithmetic nodes in the benchmark netlists, meaning that the logic minimisation phase allows the design space of potential heterogeneous blocks to be significantly extended. This is a significant improvement over the previous reported results [Smith et al. 2007].

An analysis of benchmark critical paths indicates potentially significant benefits of instantiating fused hard arithmetic components in FPGA fabrics. When implemented in a Xilinx Virtex 4 device, approximately 75% of all paths within 20% of the most critical path involve arithmetic components. For over a third of the

benchmarks 100% of these most critical paths involve arithmetic components. This means that there are significant potential system-wide benefits in terms of timing. Moreover, approximately 25% of benchmark area is devoted to arithmetic components, implying that there are also significant area savings on offer. We also note that over 50% of multipliers and approximately 40% of adders have connections to other arithmetic components, providing a strong motivation for the analysis performed in the following subsections.

5.2 Generated Cores

In order to generate the set of potential fused arithmetic cores, the subgraph extraction algorithm was run on all pairwise combinations of benchmarks. The results for the common subgraphs shown below include those subgraphs that exist in three or more benchmarks, *i.e.* do not include self-similar subgraphs, or those existing in only two benchmarks.

The maximum common subgraphs extracted from all pairwise combinations of benchmarks are shown in Table II. Included in the table are the frequencies of the graphs across the entire benchmark set and the area and delay metrics for each component when synthesised by the ASIC tools to its minimum area-delay product. To simplify the results, graphs that occur with a frequency of two or less are not presented.

Relative to common subgraph extraction for custom instruction set generation, the size of the subgraphs extracted between benchmark pairs is small. This is predominantly because such work on instruction set generation looks for common graphs within a single application (see [Biswas et al. 2005] for example). It does not make sense to do this for configurable hardware, where the fabric must be used for a number of different benchmarks. However, when compared to the self-similarity results in Table I, it is interesting to see that the approach we propose is capable of covering a large proportion of the arithmetic components within a given benchmark circuit, despite the constraint of a single output.

Compared to the results reported in [Smith et al. 2007], the combination of four additional benchmarks and the logic optimisation phase increases the number of common subgraphs from 9 to 15 (a 33% improvement). The predominant factor here is the logic minimisation phase, and can be seen by examining the breakdown of common subgraph frequencies by benchmark (Table III). The breakdown shows that, the logic optimised FIR benchmarks ('fir_3_8_8' and 'fir_24_16_16'), used in both studies, allow the discovery of pipelined adder-subtracters and more complex multiply-add blocks in Graphs 13, 14 and 15. This was not possible without the logic optimisation phase.

Table II includes the maximum output wordlength required for the generated components, as well as the minimum, maximum and mean resources required when each of the subgraphs within the benchmark set has been implemented on a Virtex 4 device. These were obtained from the synthesis of each matching substructure across all benchmarks. From these figures it is evident that when the wordlengths fit correctly, the Xilinx DSP slices can be used without the need for any fine grain logic. This is due to carry chains between DSP slices. However, the large wordlength requirements of some structures within the benchmarks necessitates the use of fine grain logic unless the cores generated by our methodology are employed.

The common subgraphs in Table II can be grouped according to type: Graphs 1-6 represent simple cascaded adder/subtracter circuits, Graphs 7-10 represent multiply accumulators and Graphs 11 and above, more complex structures extracted from the benchmark set. In the case of the cascaded adder components, the most flexible component is graph 6, and occurs 42 times. This means approximately 25% of adder components in the entire benchmark set can be supported by including this type of component. This is a significant proportion of the computational components in the benchmark set, and supports the inclusion of this type of component in FPGA architectures.

Similarly, the most flexible multiply accumulate component, graph 10, occurs 54 times. This covers approximately 42% of multiply components in the benchmark set. Given that the component can be configured to perform normal multiplication (by grounding the external input pins to the adder/subtracted), this also supports the notion of including this component. The area improvements are predominantly due to the support of longer wordlengths: 39% of the subgraphs of this type require additional slices when implemented on an FPGA, the area penalty for which is substantial.

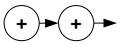
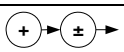
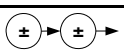
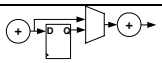
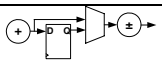
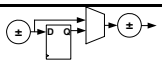
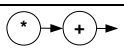
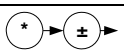
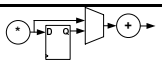
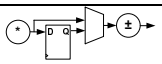
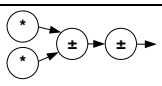
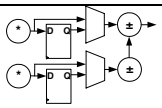
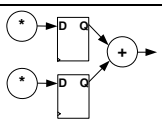
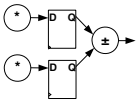
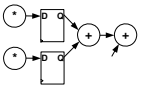
Some of the larger components provide an extra insight into the computational requirements of the benchmarks and show that these can provide significant benefits. For example, graph 14 identified by the common subgraph procedure contains a multiply accumulate as well as an additional multiplier. By using the component with an additional multiplier unit instead of a more basic multiply accumulate, it is possible to cover more of the multiplier nodes across the benchmark set (49% instead of 42%). This is at the expense of not covering several addition units.

5.3 Comparison to Virtex 4

In this section, each subgraph identified and synthesised by our framework is discussed and compared to an existing device family. Each subgraph in each benchmark was synthesised individually on the Virtex 4 platform and using the cores generated by the ASIC synthesis tools. Timing information was obtained from the estimates from the synthesis results. Area information was estimated by synthesising an ASIC core similar to that found in the Virtex 4 device. To further verify and obtain area for the slice components, we compared to area figures reported in [Beauchamp et al. 2008], in which die photos were obtained for a Virtex II device. In order to compare fairly, each subcircuit identified during the node covering phase is mapped such that the wordlength requirements and corresponding delays are appropriately evaluated. For example, a subcircuit may include a 8-bit multiply accumulate with 16 output bits. This subcircuit would not need to use all of the 36 output bits of the 18x18-bit DSP blocks in the Virtex 4, hence the tools evaluate the delays to the appropriate output bit. Similarly, our flow might identify an 18x18-bit multiply accumulate, and the evaluation of this component must only consider the propagation to output bits that are used. Thus the subcircuit speed is evaluated consistently in both ASIC and FPGA synthesis flows.

Table IV shows the relative speed and area ranges of the cascaded adder components. The geometric mean of the relative improvements seen has been used as the figure of merit. The comparison is made on a component to component basis for each subgraph identified: as relative factors are being averaged rather than absolute

Table II. Common Subgraphs structures identified with their frequency of occurrence across the entire benchmark set, with delay, size (relative to λ^2) and area-delay product for the implementation with smallest area-delay product. The maximum wordlength required, and Virtex 4 resource usages for the smallest and largest wordlength components are also given.

No.	Subgraph Structure	Freq.	Area $10^6\lambda^2$	Delay (ns)	Area × Delay $10^6\lambda^2$ns	Max output bits	Min/mean/Max Required Resources: DSPs (slices)
1		27	1.81	1.04	1.88	64	1 (0)/1.3 (0)/2 (0)
2		29	1.91	1.13	2.17	64	1 (0)/1.3 (0)/2 (0)
3		33	1.87	1.19	2.22	64	1 (0)/1.3 (0)/2 (0)
4		36	1.96	1.14	2.24	64	1 (0)/1.2 (0)/2 (0)
5		38	1.98	1.29	2.57	64	1 (0)/1.2 (0)/2 (0)
6		42	2.20	1.24	2.74	64	1 (0)/1.2 (0)/2 (0)
7		9	7.78	1.816	14.1	64	1 (0)/1.4 (3.6)/3 (16)
8		14	14.3	1.65	23.7	78	1 (0)/2 (8)/3 (16)
9		49	8.68	2.0	17.4	67	1 (0)/1.5 (13.2)/4 (50)
10		54	11.3	1.88	21.1	78	1 (0)/1.6 (13.5)/4 (50)
11		9	10.0	4.47	44.7	67	2 (0)/3.8 (45.5)/8 (67)
12		13	16.4	5.07	90.0	75	2 (0)/2.6 (11.4)/8 (128)
13		26	9.9	4.91	48.6	78	2 (0)/2.7 (13.1)/8 (67)
14		32	9.61	5.5	52.8	75	2 (0)/2.7 (13.1)/8 (67)
15		17	14.6	5.87	85.4	75	2(0)/2.5 (10.2)/8 (67)

quantities, thus the geometric mean is an appropriate average. The range given in the Table accounts for the fact we have synthesised three different components for each subgraph, each with varying time and area constraints (see Section 4.3 for details of these implementations). The minimum and maximum values show the relative differences between the generated component and synthesising on a Xilinx FPGA for the fastest and slowest individual graph from the benchmarks set. For the cascaded adder components this means that in terms of the logic speed, this component always provides an advantage over the Xilinx's lookup-based logic components used to implement this kind of functionality. However, in terms of area the component does not always provide an advantage. This is mostly because there are several adder components using a relatively small number of bits, hence the large adder, which is up to 64-bits, is too area inefficient for these smaller adders. Conversely, when all of the bits of the component are used, area savings of around 50× are possible, accompanied by speed advantages of around 5×.

Table V gives data on area and speed differences of multiply accumulate components. In this case there is much more of an obvious tradeoff in speed and area: the fastest component generated by the ASIC synthesis tool has a better geometric average than the Xilinx-synthesised version, however the corresponding geometric average in area shows that the component is larger than the Xilinx version. This is because this type of component matches the fused arithmetic resources (DSP slices) that exist on the Virtex 4 FPGA. In a similar manner to the cascaded adder components, when the wordlength requirements of the subcircuits better match the generated core, significant improvements can be made (a maximum of around 7× in both area and speed). The reason for the potential area improvements being much lower here than in the case of the cascaded adder is that the DSP-slice component has much less inherent flexibility than the fine grain slices used to implement

Table III. Summary of the common subgraphs identified in Table II by benchmark circuit.

Benchmark	Graph Number from Table II														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
boundtop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cordic	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
diffeq1	0	0	1	0	0	1	1	3	1	3	1	1	0	0	0
diffeq2	0	0	1	0	0	1	1	3	1	3	1	1	0	0	0
fir_scu_rtl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
fir_3_8_8	0	0	0	1	1	1	0	0	2	2	0	0	2	2	1
fir_24_16_16	0	0	0	8	8	8	0	0	12	12	0	0	12	12	5
frame_buff	3	3	3	3	3	3	0	0	0	0	0	0	0	0	0
iir	1	1	1	1	1	1	4	5	4	5	1	1	2	2	1
iir1	0	0	0	0	0	0	3	3	3	3	0	0	0	0	0
oc54_cpu	1	1	2	1	1	2	0	0	0	0	0	0	0	0	0
MAC1	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
MAC2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
ray_gen	14	14	15	14	14	15	0	0	12	12	0	4	4	10	4
rt_top	4	4	4	4	4	4	0	0	14	14	6	6	6	6	6
rs_decoder1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
rs_decoder2	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
Total	27	29	33	36	38	42	9	14	49	54	9	13	26	32	17

Table IV. Summary of the results (Part A): relative speed and areas of the cascaded adder components summarised in Table II, as well as the total number of routing segments used in the FPGA implementation of each subcircuit.

Metric	Common Subgraph Number (from Table II)					
	1	2	3	4	5	6
Range of Geomean Rel. Delay ($\frac{FPGA}{ASIC}$)	3.1-4.1	2.8-3.5	2.9-3.6	2.7-3.8	2.5-3.3	2.5-3.2
Min. Rel. Delay	2.6	1.98	1.07	2.15	1.91	1.65
Max. Rel. Delay	5.9	5.61	5.08	5.48	4.90	4.83
Range of Geomean Rel. Area ($\frac{FPGA}{ASIC}$)	5.6-13	4.0-14	3.1-14	6.6-16	4.3-14	3.9-13
Min. Rel. Area	2.95	0.35	0.27	2.62	0.34	0.25
Max. Rel. Area	20.03	55.8	57.1	54.4	53.6	48.4
Range of Geomean Rel. A×D ($\frac{FPGA}{ASIC}$)	22-42	15-42	11-44	28-48	15-39	11-36
Total Routing Segments Used	1323	1328	1598	2814	2819	3057

addition.

Table VI shows the advantages and disadvantages of using more complex components that were extracted from the benchmark set. These components can provide an FPGA with significant area savings of over 15× for an individual component. Again, this is when the wordlength requirements best match the generated component. An interesting characteristic not shown in the table is that some subcircuits can be constructed entirely from Xilinx DSP slices. This is due to the 3-input adder in conjunction with the carry chains between DSP slices (this is shown in Figure 1). In fact all of graphs 11-15 can be implemented using a single column of DSP slice if the wordlength requirements are sufficiently low. This removes the need for the relatively high capacitance routing resources.

Tables IV- VI also show the sum of the internal routing wires across the benchmark set used for each core. In configurable devices, the functional fabric is not the only part of the device that is used. The routing fabric also contributes a significant portion of device area. The number of segments used in routing a design determines the number of multiplexers and pass transistors and hence the substrate area used and is thus an interesting figure of merit for the potential embedded cores. Because the cores generated in our flow are chosen to support the largest wordlength possible, these wires would not be evident should the proposed cores be implemented. The routing segments used in each FPGA implementation was extracted using the Xilinx tools and isolated from the input and output routing segments. In some cases the amount of these wires is significant: for instance once we start adding pipeline registers to the cascaded adder components, a large jump in the routing segments required is seen, as the adders can no longer be fit into a single slice without some sort of retiming. However, if Graph 11 is considered, only a relatively small number of routing segments is used for such a relatively complex core. This is due to the efficient carry chains implemented in the Xtreme DSP slices used in the Virtex 4 device.

Table V. Summary of the results (Part B): relative speed and areas of the multiply accumulate components summarised in Table II, as well as the total number of routing segments used in the FPGA implementation of each subcircuit.

Metric	Common Subgraph Number (from Table II)			
	7	8	9	10
Range of Geomean Rel. Delay ($\frac{FPGA}{ASIC}$)	1.41-1.92	2.43-2.97	0.99-1.28	1.64-1.99
Min. Relative Delay	0.69	0.76	0.62	0.67
Max. Relative Delay	7.08	7.07	5.8	7.13
Range of Geomean Rel. Area ($\frac{FPGA}{ASIC}$)	0.81-1.36	0.66-1.19	0.43-0.92	0.78-1.41
Min. Relative Area	0.55	0.28	0.38	0.36
Max. Relative Area	5.13	2.79	4.6	7.11
Range of Geomean Rel. A×D ($\frac{FPGA}{ASIC}$)	1.19-2.01	1.97-3.0	0.53-0.92	1.55-2.41
Total Routing Segments used	242	716	4074	4548

Table VI. Summary of the results (Part C): relative speed and areas of components with more than two arithmetic blocks, as summarised in Table II, as well as the total number of routing segments used in the FPGA implementation of each subcircuit.

Metric	Common Subgraph Number (from Table II)				
	11	12	13	14	15
Range of Geomean Rel. Delay ($\frac{FPGA}{ASIC}$)	1.76-2.59	1.43-2.04	0.99-1.55	0.86-1.36	0.98-1.43
Min. Relative Delay	1.21	0.76	0.81	0.42	0.41
Max. Relative Delay	3.44	3.16	3.31	3.11	3.3
Range of Geomean Rel. Area ($\frac{FPGA}{ASIC}$)	1.97-4.96	1.17-3.13	0.70-2.04	0.73-2.30	0.63-1.5
Min. Relative Area	0.63	0.41	0.5	0.48	0.47
Max. Relative Area	1.59	16.4	15.1	15.5	11.6
Range of Geomean Rel. A×D ($\frac{FPGA}{ASIC}$)	5.1-9.04	2.39-4.49	1.08-2.02	1.0-1.98	0.25-0.43
Total Routing Segments used	670	4893	3058	4180	5874

6. CONCLUSION

In this paper, we have presented a methodology for extracting commonly occurring patterns in circuit netlists. This has led to the quantification of potential benefits in terms of area, delay and configurable routing segments of a set of arithmetic components. The reported improvements indicate that there are arithmetic cores that have the potential to improve FPGA logic density and performance by significant amounts. There is potential for significant future work. For example, we intend to further address the system-level benefits of the components of new embedded silicon cores: the cost associated with routing to and from such components and how this affects the performance of the entire system is an important factor. We also intend

to extend our benchmark set to incorporate a larger domain through the use of publicly available benchmarks such as [Extensible, Programmable and Reconfigurable Embedded Systems Group]. Further interesting study could also examine multiple input multiple output (MIMO) graphs in order to find more complex structures within the benchmark set.

Acknowledgment

This work has been funded by the EPSRC (UK) under grant numbers EP/C549481/1 and EP/E00024X/1.

REFERENCES

Altera Corporation 2005a. Stratix device handbook, volume 2.

Altera Corporation 2005b. Stratix II device family data sheet.

Aravind, D. and Sudarsanam, A. 2005. High level - application analysis techniques & architectures - to explore design possibilities for reduced reconfiguration area overheads in fpgas executing compute intensive applications. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, Washington, DC, USA, pp. 158.1. IEEE Computer Society.

Beauchamp, M. J., Hauck, S., Underwood, K. D., and Hemmert, K. S. 2008. Architectural modifications to enhance the floating-point performance of fpgas. *IEEE Transactions on VLSI Systems 16,* 2, 177–187.

Biswas, P., Banerjee, S., Dutt, N., Pozzi, L., and Ienne, P. 2005. Isegen: generation of high-quality instruction set extensions by iterative improvement. In *Design, Automation and Test in Europe*, Volume 2, pp. 1246–1251.

Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., and Vento, M. 2002. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, London, UK, pp. 123–132. Springer-Verlag.

Cadence Design Systems Inc. 2005a. Encounter^{TM} user guide, product version 5.2.

Cadence Design Systems Inc. 2005b. Using Encounter^{TM} RTL Compiler, Product Version 5.2.

Chowdary, A., Kale, S., Saripella, P. K., and Sehgal, N. K. G. R. K. 1999. Extraction of functional regularity in datapath circuits. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 18,* 9, 1279–1296.

Compton, K. and Hauck, S. 2007. Automatic design of area-efficient configurable asic cores. *IEEE Transactions on Computers 56,* 5, 662–672.

Cone, M. M., Venkataraghven, R., and McLafferty, F. W. 1977. Molecular structure comparison program for the identification of maximal common substructures. *Journal of the American Chemistry society 99,* 23, 7668–7671.

Cong, J. and Ding, Y. 1994. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 13,* 1, 1–13.

Cong, J., Fan, Y., , Han, G., and Zhang, Z. 2004. Application-specific instruction generation for configurable processor architectures. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, New York, NY, USA, pp. 183–189. ACM.

Extensible, Programmable and Reconfigurable Embedded Systems Group. Express Benchmark Suite, http://express.ece.ucsb.edu/benchmark/.

Jamieson, P. and Rose, J. 2005. A verilog RTL synthesis tool for heterogeneous FPGAs. In *Field Programmable Logic and Applications*, pp. 305–310. IEEE.

Kastner, R., Kaplan, A., Ogrenci-Memik, S., and Bozorgzadeh, E. 2002. Instruction generation for hybrid reconfigurable systems. *ACM Transactions on Design Automation of Electronic Systems 7,* 4, 605–627.

Kuon, I. and Rose, J. 2007. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26,* 2, 203–215.

Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R., and Sangiovanni-Vincentelli, A. 1992. SIS: A system for sequential circuit synthesis. Technical report, EECS Department, University of California, Berkeley.

Smith, A., Constantinides, G. A., and Cheung, P. Y. K. 2007. Fused-arithmetic unit generation for reconfigurable devices using common subgraph extraction. In *International Conference on Field Programmable Technology*, pp. 105–112. IEEE.

STMicroelectronics 2006. 90nm CMOS090 Design Platform.

Williams, S. 1999. ICARUS Verilog, http://www.icarus.com/eda/verilog/.

Xilinx Inc. 2004a. White paper - xilinx virtex-4 revolutionizes platform FPGAs.

Xilinx Inc. 2004b. XtremeDSP Design Considerations User Guide.

Xilinx Inc. 2006. Xilinx integrated software environment (ISE), version 8.2i.