Fused-Arithmetic Unit Generation for Reconfigurable Devices using Common Subgraph Extraction

Alastair M. Smith, George A. Constantinides and Peter Y. K. Cheung
Imperial College, London SW7 2BT, U.K.
{alastair.smith,g.constantinides,p.cheung}@imperial.ac.uk

***Abstract*— To complement the flexible, fine-grain logic in Field Programmable Gate Arrays (FPGAs), configurable hardware devices now incorporate more complex coarse-grain components such as memories, embedded processing units and fused-arithmetic units. These components provide speed and density advantages due to the specialised logic and fixed interconnect. In this paper, a methodology is presented to automatically propose and explore the benefits of different types of fused arithmetic units for configurable devices. 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. We report improvements of up to $3.3\times$ in speed and $19.7\times$ in area for the average improvement of particular silicon cores identified by our approach when compared to implementation of the same subcircuits implemented in a commercial mixed-granularity FPGA in a comparable 90nm technology. The average improvements across all embedded cores identified by our approach are $1.67\times$ and $5.55\times$ when designing the ASIC cores for fastest speed performance.**

I. Introduction

For embedded systems, reconfigurable devices provide designers with high-throughput, 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 wide-spread use of reconfigurable architectures for high throughput computation, there have been several advances in reconfigurable chip design for this domain, particularly evident in Field Programmable Gate Arrays (FPGAs).

Modern FPGAs consist of a variety of different resource types to implement various functionality. 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 logic fabric can be inefficient when compared to gates for particular logic functions. More critically, in order to connect the LUTs, a flexible routing fabric is required, typically using programmable switches to arbitrarily connect blocks, 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 [1], [2].

This paper focuses on fused-arithmetic components in FPGAs. An example of a commercial fused-arithmetic unit is the Xtreme Slice [3], evident in Xilinx's Virtex 4 family of FPGAs. A simplified structure of the Xtreme Slice is shown in Figure 1. One of the central features of this embedded core is the flexibility that is built in to provide support for a wide variety of applications via configuration multiplexers. 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.
- 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 II related work is discussed. The problem of common subgraph extraction for fused-arithmetic component generation is formally defined in Section III. Section IV details the design flow and algorithm used to extract commonly occurring arithmetic subcircuits. Section V presents the methodology that is employed to synthesise and compare the commonly

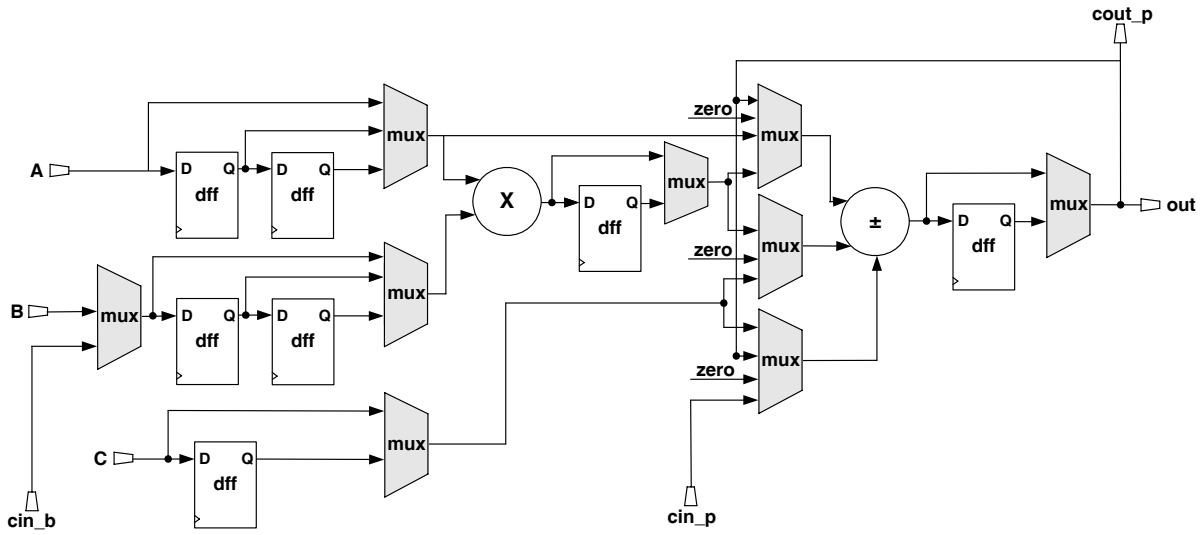


Fig. 1. A simplified view of the Xilinx Xtreme DSP Slice [3]. The MUX select lines are driven by configuration bits.

occurring patterns. Comparisons between subcircuits implemented in the generated silicon cores and those implemented in a commercially available device are presented in Section VI. Finally, conclusions are drawn in Section VII.

II. 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 [4]. 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 [5] and custom instruction-set generation [6], [7], [8]. 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 [7], 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 [8], 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.

There have also been efforts to use common subgraph detection methods for the purpose of discovering hard-cores for FPGAs [9], 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 analysis 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 [10] architectures are generated for a specific set of benchmarks, and methods are employed to reduce the number of multiplexers and connecting wires. Similar to [10], 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 more 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 appropriate sized functional blocks. This is a well researched topic, exemplified by FlowMap [11] and SIS [12]. With the

modern advances in FPGA technology, synthesis now has to 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 [13], 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 then used to determine which benchmark structures can then be matched by the common subgraphs.

III. 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*.

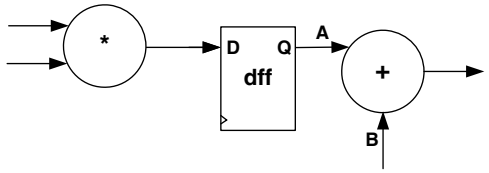
We use word-level netlists as formal representations of benchmark circuits after parsing and elaboration of Verilog, where the node set corresponds to word-level arithmetic operations such as addition (a *type*). The edge set corresponds to the flow of data between these operations. We associate an order with the edge set to ensure a unique representation of unique inputs, *e.g.* 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 relation 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.

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 $\forall u \forall v\,[(u,v) \in E \wedge u \in V' \wedge v \in V' \Leftrightarrow (u,v) \in E']$, and such that the order on E is preserved in E' for inedges from nodes with types in $\mathcal{T}^{NC}$.

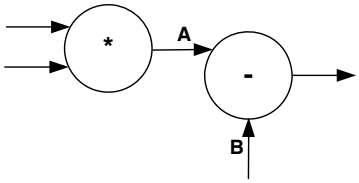
This definition captures the concept of 'part of a computation', while the final conditions ensure that (i) if two nodes are part of a subgraph, then so are the relevant edges, and (ii) if the order of inputs to a computation is important, then it is preserved.

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

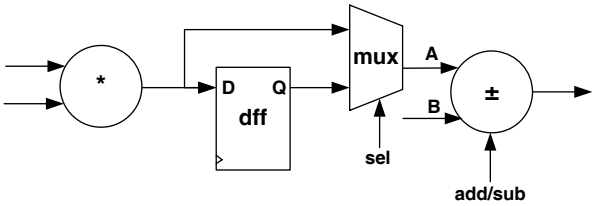
Isomorphism is useful in this context, because if two word-level netlists are isomorphic, they compute the same function.



(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 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. 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) $T' \subseteq T$.

We thus 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.

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 T(v_i) = T(v_j) then
5:        if ISMATCH(v_i, v_j) then
6:          add nodes to G'
7:          FAST_MATCH(G_1, G_2, G')
8:        end if
9:      end if
10:   end for
11: end for
```

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 ports i on node r_1(u') do
3:      u_1 = node on port i of r_1(u')
4:      for all ports j on node r_2(u') do
5:        u_2 = node on port j of r_2(u')
6:        if ISMATCH(u_1, u_2) then
7:          add node to G' on appropriate port of u'
8:          FAST_MATCH(G_1, G_2, G')
9:          remove f^{-1}(u_1) and its connections from G'
10:       end if
11:     end for
12:   end for
13: end for
```

IV. Common Subgraph Extraction Algorithm

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. For this reason, a further constraint on the common subgraph extraction algorithm is that the common subgraph should have only one output.

A. Tool Flow and Common Subgraph Extraction Algorithm

A combination of the publicly available tools Icarus Verilog [14] and ODIN [13] 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 common subgraph extraction algorithm used for this work is based on one in [15], 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 employedto 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 to find two similar nodes (multipliers and adders *or* subtracters). 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 describes the recursive algorithm for adding nodes to the common subgraph. The functions $r_1(u')$ and $r_2(u')$ return the nodes from benchmark netlists G_1 and G_2 respectively which have been mapped onto common subgraph node u'. Thus the recursive algorithm attempts to add nodes to the common subgraph by examining the inputs to the nodes related from each circuit netlist. Each arithmetic node in the netlist produced by ODIN, once contracted, has two ports thus it is important to attempt a match for each combination of inputs. The ordering of inputs to subtract operator must be dealt with correctly. In the common subgraph extraction add and subtract nodes are allowed to be matched to create a component with a configuration port.

In Algorithms 1 and 2, the 'ISMATCH' function first has to determine whether a component can be added to the graph. This has to ensure that adding a combined node satisfies the problem constraints. 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.

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.

B. Technology Mapping: Node Covering

An important aspect of the methodology developed is the technology mapping. Once there is a library of common subgraphs extracted from all pairwise combinations in the benchmark set, an important consideration for the device architect is how frequent these graphs are. The more frequent the graphs are, the better they are as a candidate to be used in the configurable fabric. The frequency and size of the components is also critical in deciding the size of the component to be included on the device.

The problem of technology mapping is essentially one of node covering, and has been addressed in previous work [13]. The problem of node covering is shown graphically in Figure 3. In this paper, the problem is dealt with in a straightforward manner: a complete enumeration of potential mappings is made, with the mapping that covers the most nodes being the one that is chosen. The input and output word-lengths of the components in this maximum subgraph frequency mapping are then considered for synthesis of each of the common subgraphs extracted.

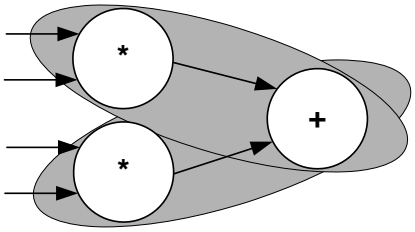


Fig. 3. Potential graph coverings for technology mapping part of a benchmark DFG where the common subgraph examined is that of Figure 2(c).

V. 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 [16]. 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. The synthesis routine to ascertain speed sandwiches the benchmark subcircuit between registers and observes the register-to-register delay, ensuring that pad delays and routing delays from pads to the components are minimised. 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 [17] and SoC Encounter [18], 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 90nnm library [19] was used at the appropriate voltage (1.2V). For each common subgraph, ASIC synthesis is performed three times to give a range of cores with different speed and area. First, an attempt to force the component to operate at 2Ghz is made, which is an unreasonable constraint, but provides information about the fastest the component can operate. The component is then constrained to work at a half and a third of the maximum operating frequency found.

VI. Results

A. Generated Cores

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 [13], spanning a variety of application domains such as DSP and scientific computing. Note that only a subset of the designs are used, as several of the benchmarks are essentially replicated with differences only in the memory components. This paper focuses on benchmarks that have arithmetic components: various statistics of each benchmark presented in Table I. The common subgraph extraction algorithm was performed pairwise on each combination of two benchmarks.

The benchmarks in Table I marked with an asterisk were found to have no common subgraphs with more than one arithmetic component, and are hence not considered in further results. In the case of the FIR filters, combinatorial logic exists between arithmetic nodes in the benchmark netlists, meaning that fused-arithmetic components cannot be discovered for these circuits.

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

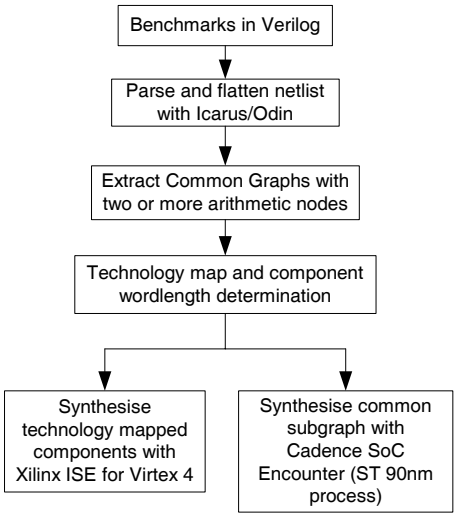


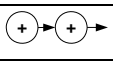
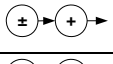
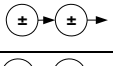
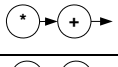
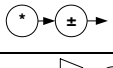
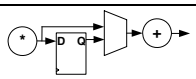
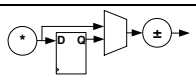
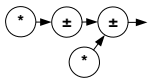
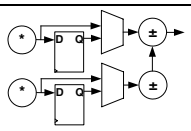
Fig. 4. The design flow.

TABLE I
Summary of the arithmetic components that exist in each benchmark circuit.

Benchmark	Number of Add/Subs	Number of Multipliers
diffeq1	4	5
diffeq2	4	5
frame_buff	17	0
iir	6	5
iir1	16	5
ray_gen	47	18
reed_sol_decoder1	6	13
reed_sol_decoder2	16	9
boundtop*	47	0
cordic*	49	0
fir_scu_rtl*	16	17
fir_3_8_8*	3	4
fir_24_16_16*	24	25

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.

Graph No.	Common Subgraph Structure	Freq.	Area $10^6\lambda^2$	Delay (ns)	Area × Delay $10^6\lambda^2$ns
1		18	0.47	2.19	1.05
2		20	0.51	2.41	1.22
3		23	0.84	2.17	1.82
4		9	2.45	4.42	10.8
5		13	4.49	2.51	11.2
6		21	4.52	2.65	12.0
7		25	4.76	2.55	12.2
8		3	5.42	4.29	23.3
9		9	4.52	5.55	25.1

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 are not presented. The graphs can be grouped according to type: Graphs 1-3 in Table II represent simple cascaded adder/subtracter circuits, Graphs 4-7 represent multiply accumulators and graphs 8 and 9 more complex structures extracted from the benchmark set.

The first point of interest concerns the area-flexibility trade-off. The first three graphs in Table II show an increase in flexibility (number of structures that can be mapped onto them). The cost of this increase comes from two sources: the requirement of an adder/subtracter component rather than a simple adder, and the increasing word-length required for the component (32-bits).

Graphs 6, 7 and 9 represent common subgraphs that have been identified with optional pipeline registers. The advantages are quite simple: considerably more structures in the benchmark set can be mapped onto these components when the optional pipelining is identified. This comes with a certain penalty on area and delay (less than a 4% area overhead for the fastest speed multiply accumulate cores).

B. Comparison to Virtex 4

In order to compare the components generated for the common subgraphs identified with those in a commercial device, it is important to consider each technology mapped subcircuit separately. Each occurrence of a given common subgraph across the set of benchmark netlists will have differing delays according to the wordlength requirements. Thus each subcircuit is technology mapped onto a Virtex 4 and onto the generated silicon core and the two implementations compared in terms of speed and silicon area. The results are presented in Table III.

As an additional study, the number of routing segments used by the FPGA implementation of each subcircuit was evaluated. The effects of routing are inherent in the delay figures reported by the FPGA synthesis tools, however, given that the amount of area devoted to the configurable routing segments is large when compared to fixed routing structures within the embedded cores, it is important to consider this also. For this study, the number of routing segments calculated considers only those in the general routing fabric *i.e.* not configurable routing wires in carry chains. These results are also shown in Table III.

As previously mentioned, the silicon cores have been generated under three different area and time constraints using the ASIC tools in order to give a range of area and speed values for these components. The results table presents the range of these values as the geometric mean of the speed of the ASIC implementation relative to the speed of the subcircuit implemented in Virtex 4, with the maximum and minimum individual relative delay values across all three implementations also given. Similarly, area results are given as relative improvements of the generated ASIC core over a Virtex 4 implementation. Also included in the table is the range of the geometric means of the relative area-delay product across the three different speed grades.

Table III shows that on average, the relative delay of any of the ASIC cores improves over an implementation of the same subcircuit in a Virtex 4 from 0.84-2.19×. Clearly there is not always an improvement in delay, which is for several reasons. One reason is the full custom design of the DSP slice: allowing the component to be tuned and optimised by hand is likely to provide better gains than automated ASIC synthesis. Component word-length is another obvious contributor to this: take for example, both diffeq benchmarks, which require 32-bit multipliers whereas the ray_gen benchmark requires 8 and 16-bit multipliers. For these examples, the FPGA fabric will require 3 Xtreme DSP slices for the 32-bit multiplier (not all output bits are used, otherwise this value would be 4) and only one DSP slice for the 8 and 16-bit multipliers, whereas as they will all require only one of the generated cores; the fine-grain FPGA fabric can be configured to be better customised to the word-length requirements of each individual subcircuit due to the smaller word-lengths of the Xtreme DSP slice and associated carry chains.

The cascaded adder components (Graphs 1-3 in Tables II

TABLE III

SUMMARY OF THE RESULTS: THE FREQUENCIES, AND RELATIVE SPEED AND AREAS OF THE IDENTIFIED COMPONENTS SUMMARISED IN TABLE II, AS WELL AS THE TOTAL NUMBER OF ROUTING SEGMENTS USED IN THE FPGA IMPLEMENTATION OF EACH SUBCIRCUIT.

Benchmark	Common Subgraph Number (from Table II)								
	1	2	3	4	5	6	7	8	9
diffeq1	0	0	1	1	3	1	3	1	1
diffeq2	0	0	1	1	3	1	3	1	1
frame_buff	3	3	3	0	0	0	0	0	0
iir	1	1	1	3	3	3	3	1	1
iir1	0	0	0	4	4	4	4	0	0
ray_gen	14	14	15	0	0	12	12	0	6
reed_sol_decoder1	0	1	1	0	0	0	0	0	0
reed_sol_decoder2	0	1	1	0	0	0	0	0	0
Range of Geometric mean of of Relative Delay ($\frac{FPGA}{ASIC\ core}$)	0.94-1.93	0.88-1.68	1.15-1.70	0.43-1.07	0.61-1.56	0.31-0.84	0.34-1.04	1.18-3.26	0.81-2.19
Min. Relative Delay	0.74	0.69	0.93	0.18	0.19	0.18	0.16	0.72	0.48
Max. Relative Delay	2.3	2.09	2.80	4.07	3.96	3.94	3.73	4.40	4.26
Range of Geometric mean of Relative Area ($\frac{FPGA}{ASIC\ core}$)	19.7-49.8	10.7-42.9	6.91-26.1	1.73-5.80	2.28-8.56	1.33-4.39	1.55-5.51	3.98-12.4	1.76-5.47
Min. Relative Area	10.9	1.31	0.82	1.18	1.04	1.13	1.03	1.27	1.2
Max. Relative Area	80.5	84.2	49.8	21.9	21.4	20.57	20.1	21.9	20.6
Range of Geometric mean Relative A×D ($\frac{FPGA}{ASIC\ core}$)	42.7-66.6	44.2-105	13.2-33.9	1.85-3.28	3.55-6.33	1.12-1.86	1.61-2.83	12.9-18.7	3.49-5.37
Total Routing Segments used in FPGA Implementation	314	314	400	242	724	242	724	670	670

and III), provide a significant range of improvements in delay and area. The fast component generated for each of these subcircuits gains on average 1.93, 1.68 and 1.7× in delay. An important result not shown by the table is that for each cascaded adder/subtracter component, the component with the smallest area-delay product gains in both area and delay over an FPGA implementation for all subcircuits mapped to the component. This is despite some of the benchmark components having particularly small word-lengths. The minimum area-delay product improvement of 3.8×, showing that the area-delay product is always improved with this type of component.

One of the particularly important advantages of cascaded adder/subtracter components is the elimination of programmable routing wires between components. The advantages are partly evident in the reported improvements in circuit delay: capacitance heavy configurable routing segments in the FPGA implementation are one of the sources of the relatively slow speed of the Virtex 4 implementation. Furthermore, eliminating routing wires also provides gains in terms of the substrate area used, due to the reduction of pass transistors and buffers used in the programmable switches.

Graphs 8 and 9 are more complex components that have been identified by the framework. There is a significant drop in the improvements offered when optional pipelining is added. This is partly because of the increased complexity of the proposed ASIC core. However, the primary reason for this is the efficiency of the DSP slice at implementing the functionality of the additional structures that can be mapped: in the Virtex 4 implementation, DSP carry chains are employed to negate the use of the relatively inefficient fine-grain fabric. This is emphasised by considering the fact that only two of the mapped structures use fine-grain configurable routing wires.

Across all subgraphs identified improvements in delay and speed have been reported. The average improvement in delay across all identified subcircuits when compared to FPGA implementation is $1.67\times$ when designing the ASIC core for fastest speed, and the average improvement in area for the same silicon cores is $5.55\times$. The gains in area-delay product also highlight the advantages of the potential embedded silicon cores.

VII. CONCLUSION

In this paper, we have presented a methodology for extraction of 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. In future work we intend to 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 effects the performance of the entire system is an important factor.

ACKNOWLEDGMENT

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

REFERENCES

[1] Altera Corporation, San Jose, *Stratix II Device Family Data Sheet*, 2005.

[2] Xilinx Inc., San Jose, *White Paper - Xilinx Virtex-4 Revolutionizes Platform FPGAs*, 2004.

[3] ——, *XtremeDSP Design Considerations User Guide*, 2004.

[4] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[5] A. Chowdary, S. Kale, P. K. Saripella, and N. K. G. R. K. Sehgal, "Extraction of functional regularity in datapath circuits." *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 18, no. 9, pp. 1279–1296, 1999.

[6] M. M. Cone, R. Venkataraghven, and F. W. McLafferty, "Molecular structure comparison program for the identification of maximal common substructures."

[7] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *ACM International Symposium on Field Programmable Gate Array*, 2004.

[8] R. Kastner, A. Kaplan, S. Ogrenci-Memik, and E. Bozorgzadeh, "Instruction generation for hybrid reconfigurable systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 605–627, 2002.

[9] D. Aravind and A. Sudarsanam, "High level - application analysis techniques and architectures - to explore design possibilities for reduced reconfiguration area overheads in FPGAs executing compute intensive applications," in *IEEE International Parallel and Distributed Processing Symposium*, 2005.

[10] K. Compton and S. Hauck, "Automatic design of area-efficient configurable asic cores," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 662–672, 2007.

[11] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs."

[12] "SIS: A system for sequential circuit synthesis," EECS Department, University of California, Berkeley, Tech. Rep., 1992.

[13] P. Jamieson and J. Rose, "A verilog RTL synthesis tool for heterogeneous FPGAs," in *IEEE Field Programmable Logic and Applications*, 2005.

[14] "ICARUS Verilog http://www.icarus.com/eda/verilog/."

[15] H. Bunke, P. Foggia, G. Corrado, C. Sansone, and M. Vento, "A comparison of algorithms for maximum common subgraph on randomly connected graphs," in *Syntactical and Structural Pattern Recognition and Statistical Pattern Recognition*, 2002.

[16] Xilinx Inc., San Jose, "Xilinx integrated software environment (ISE), version 8.2i," 2006.

[17] Cadence Design Systems Inc., San Jose, *Using Encounter™ RTL Compiler, Product Version 5.2*, 2005.

[18] ——, *Encounter™ User Guide, Product Version 5.2*, 2005.

[19] STMicroelectronics, Geneva, "90nm cmos090 design platform," May 2006.