Design Space Exploration for Field Programmable Compressor Trees

Seyed Hosein Attarzadeh Niaki[1] Alessandro Cevrero[2] Philip Brisk[3] Chrysostomos Nicopoulos[3]
Frank K. Gurkaynak[4] Yusuf Leblebici[2] Paolo Ienne[3]

[1] Royal Institute of Technology
School of Information and Communication Technology
Stockholm, Sweden
shan2@kth.se

Ecole Polytechnique Federale de Lausanne
[2]School of Engineering
[3]School of Computer and Communication Sciences
Lausanne, Switzerland, CH-1015
{first_name.last_name}@epfl.ch

[4] Swiss Federal Institute of Technology, Zurich
Microelectronics Design Center
Zurich, Switzerland, CH-8092
kgf@ee.ethz.ch

ABSTRACT

The *Field Programmable Compressor Tree (FPCT)* is a programmable compressor tree (e.g., a Wallace or Dadda Tree) intended for integration in an FPGA or other reconfigurable device. This paper presents a *design space exploration (DSE)* method that can be used to identify the best FPCT architecture for a given set of arithmetic benchmark circuits; in practice, an FPGA vendor can use the design space exploration to tailor the FPCT to meet the needs of the most important benchmark circuits of the vendor's largest-volume clients. One novel feature of the DSE is the introduction of a metric called *I/O utilization*; we found that I/O utilization has a strong correlation with both the critical path delay and area of the benchmark circuits under study. Pruning the search space using I/O utilization allowed us to reduce significantly the number of FPCTs that must be synthesized and evaluated during the DSE, while giving high confidence that the best architectures are still explored. The DSE was applied to seven small-to-medium range benchmark circuits; one FPCT architecture was found that was 30% faster than the second best in terms of critical path delay, and only 3.34% larger than the smallest.

Categories and Subject Descriptors

B.7.1 [**Hardware**]: Integrated Circuits – *gate arrays*.

General Terms

Performance.

Keywords

Field Programmable Compressor Tree (FPCT), Design Space Exploration (DSE).

CASES'08, October 19–24, 2008, Atlanta, Georgia, USA.

1. INTRODUCTION

FPGA performance is currently lacking for arithmetic circuits. In particular, FPGAs cannot exploit one of the fundamental results of computer arithmetic: the use of carry-save (based) addition to efficiently add $k > 2$ integers. This representation was introduced by Wallace [21] and Dadda [9] in the context of parallel multiplier design in the 1960s. The addition of two integers requires the use of a *carry propagate adder (CPA)*, e.g., a ripple-carry or parallel-prefix adder, whose critical path delay is from the carry-in to the carry-out bit. The naïve method to add $k > 2$ integers is to use an *adder tree*, i.e., a binary tree of CPAs; a *compressor tree* (e.g., a Wallace or Dadda Tree) is a circuit that uses carry-save addition to add $k > 2$ integers: the output is two integers, *S (sum)* and *C (carry)* such that $S+C$ is the sum of the k integers. Thus, a CPA is only required to add S and C, rather than at every level of the tree.

Multi-operand integer addition occurs in a wide variety of applications including, but not limited to, FIR filters [14], video coding [8], and 3G wireless base station channel cards [17]. Verma and Ienne [20] have introduced a set of data flow transformations that can expose opportunities to exploit the carry-save representation. First, each multiplication operation is decomposed into a partial product generator, compressor tree, and CPA. A sequence of sorting rules is applied to the flowgraph to reorder the operations such that disparate CPAs are merged with one another and with other compressor trees. Each subsumption of a CPA into a larger compressor tree replaces the carry-in to carry-out delay with a smaller delay due to a slightly enlarged compressor tree.

These transformations automate optimization steps that expert designers have been applying manually for years. The judicious exposition and exploitation of compressor trees is one area where ASIC designs have a significant advantage over processors and FPGAs. Multi-input addition is performed serially in a single-issue processor or use a partially serialized adder tree in a multi-issue VLIW or superscalar (partial serialization occurs if the number of integers to add exceeds the register file bandwidth). A secondary drawback is that the bitwidth of all of the adders is fixed, based on the instruction set of the processor.

Now, let us turn our attention to FPGAs which contain embedded hard multipliers and DSP blocks. The bitwidths of both hard

multipliers and DSP blocks are fixed, creating a potential mismatch between the needs of the application and the solution provided by the target architecture. Additionally, neither the multipliers nor the DSP blocks expose their compressor trees directly to the user. The user cannot bypass the partial product generator in the multiplier to directly access the compressor tree; likewise, the DSP blocks offer efficient multiply-accumulate (MAC) functionality, but only in a highly pipelined and sequentialized mode. Thus, despite the fact that multi-input addition and multiplication are based on the same circuitry, the embedded cores inside an FPGA cannot be used for both.

Therefore, the user must realize multi-input addition using the general logic of the FPGA. Modern high-performance FPGAs contain logic blocks that can be configured as ternary (3-input) carry-propagate adders; fast carry chains connect adjacent logic blocks (to one another), bypassing the costly programmable routing network. Support for ternary addition was first integrated into the *Adaptive Logic Module (ALM)* of *Altera's Stratix*-series FPGAs, starting with the *Stratix II* [13]; *Xilinx* integrated similar functionality into the *Configurable Logic Block (CLB)* of their *Virtex*-series devices, starting with the *Virtex 5* [22]. One of the key selling points of this logic block architecture was that the number of logic levels in a ternary adder tree is less than the number of logic levels in a binary adder tree [2].

For a long time, it was thought that adder trees were faster than compressor trees on FPGAs due to the presence of dedicated adders and carry chains within the logic blocks. Parandeh-Afshar et al. [15, 16] have since shown that this is not true for modern high-performance FPGAs whose logic blocks contain 6 (rather than 4, previously) inputs. Parandeh-Afshar et al. developed methods to synthesized compressor trees whose critical path delays were less than the delays of the adder trees; however, the programmable routing network still contributed significantly to the overall delay and the ternary adder trees required fewer LUTs.

In a previous paper [7], we introduced the *Field Programmable Compressor Tree (FPCT)*[1], a programmable compressor tree intended for integration in an FPGA or other reconfigurable device. The FPCT is distinguished from the DSP blocks and hard multipliers that are embedded in modern high-performance FPGAs in two respects: (1) The FPCT can be programmed to match precisely the bitwidth of the input operands, and (2) the FPCT exposes its compressor tree directly to the user, allowing faster multi-operand addition than the compressor trees synthesized by Parandeh-Afshar et al. [15, 16].

A vendor who sells FPGAs or other reconfigurable devices will want to tailor the design of their FPCTs to the most important benchmark circuits of their largest clients—especially if the company has a small client base. To meet this need, this paper introduces a *design space exploration (DSE)* method that determines the best FPCT architecture for a set of benchmarks deemed by the vendor to be of sufficient importance. An FPCT can be described in terms of three parameters, which are described in Section 3. T|he DSE enumerates the different legal combinations of these three parameters, each of which corresponds to a different FPCT; each FPCT is synthesized, and each application is then mapped onto it. From the mapping, we can determine the area and delay for each benchmark. The areas and delays are averaged, yielding a Pareto curve in terms of area and delay. The Pareto optimal solutions are then presented to the vendor, who can then select the architecture with the guarantee of Pareto optimality.

To speed up the DSE, we introduce a metric called *I/O utilization*, which can be computed for each FPCT prior to synthesis. We have observed a strong correlation between I/O utilization and the delay and area observed for each benchmark following synthesis. To prune the design space, the I/O utilization is computed for each benchmark on each FPCT that has been enumerated; only those architectures for which at least one benchmark has a high I/O utilization are synthesized and evaluated directly. Due to this correlation, the user can prune the search space while retaining high confidence that the DSE will still explore the best FPCTs.

2. FPCT ARCHITECTURE

This section summarizes the key design points of the FPCT architecture and explains which parameters are varied by the DSE in the following section.

The basic unit of computation in an FPCT is called a *Compressor Slice (CSlice)*; an FPCT is a 1-dimensional array of CSlices. A CSlice takes as its input a set of bits to be summed; it sums these bits, and produces one (or more) output bits representing the sum. Additionally, each CSlice propagates carry-out bits to subsequent CSlices, and receives carry-in bits from it preceding CSlices. The mapping process, which synthesizes an instance of multi-input addition or partial product reduction on an FPCT maps all of the input bits onto a contiguously set of CSlices. Details on the architecture and mapping process can be found in a prior paper by Cevrero et al. [7].

Fig. 1 illustrates the basic CSlice architecture. Three parameters of the CSlice are varied by the DSE, and are highlighted in gray: the *First Counter Size (FCS)*, the *Generalized Parallel Counter (GPC) Configuration Circuit (GPCCC)*, which subsumes the *Input Configuration Circuit (ICC)*, and the *Maximum Output Rank Configuration (MORC)*. Two other modules shown in Fig. 1—the *Chain Interrupt Configuration Circuit (CICC)* and *Output Multiplexing Circuit (OMC)* are necessary for correct operation, but can be derived deterministically from the FCS, GPCCC/ICC, and MORC. The remainder of this section describes these three parameters and modules in detail.

2.1 First Counter Size (FCS)

An *m:n* (parallel) counter is a circuit that takes in *m* input bits, counts the number of inputs that are set to *'1'*, and outputs the result as an unsigned *n*-bit binary integer in the range *[0, m]*; it follows that $n = \lceil log_2(m+1) \rceil$.

Parallel counters are a fundamental building block for compressor trees [19]. In the FPCT, each CSlice contains a vertical chain of counters in descending order of size; the pattern is as follows: the number of output bits of the following counter is equal to the number of input bits of the preceding counter, followed by a CPA. In Fig. 1, the chain is: *{31:5*, *5:3*, *3:2}*; in general, the size of each counter in the chain is deterministic once the FCS is given.

[1] In our prior work [6, 7], the name of the FPCT was "Field Programmable Counter Array" (FPCA); ref. [6] describes a preliminary version that was revised significantly in ref. [7].

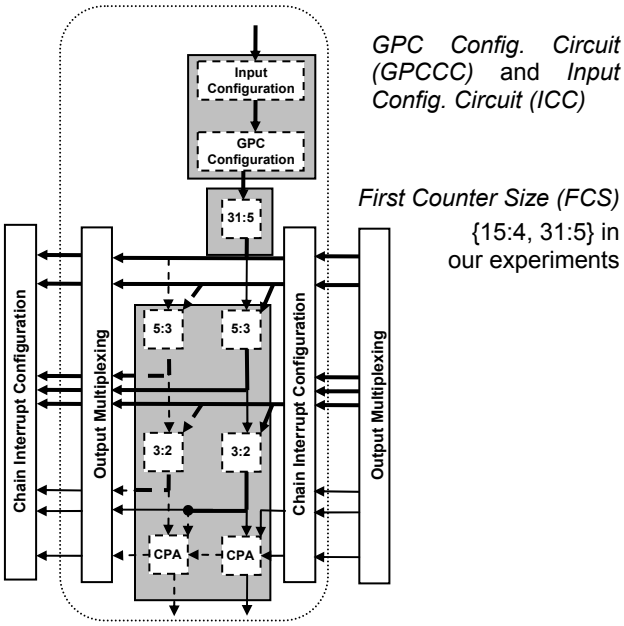

Figure 1.

The CSlice architecture template [7, Fig. 4(e)], which is characterized by 3 parameters.

Increasing the FCS increases the input bandwidth of the CSlice (as well as the FPCT, as all CSlices are identical). The FCS, however, is the largest component in the CSlice, so increasing its size may significantly impact the area of the FPCT.

The *Compression Ratio* of an *m:n* counter is the ratio *m/n* of the number of input to output bits; for a fixed number of output bits *n*, *m/n* is maximal when $m = 2^n - 1$. For example, the compression ratio of a *7:3* counter is *2.33*, while that of a *6:3* counter is *2*. As the goal of the FPCT is to compress a large number of input bits down to two per column (which are then added by the CPA), counters with higher compression ratios are the most effective. Our DSE considered *15:4* and *31:5* counters for our FCS; for the benchmarks considered in this study, *63:6* counters were simply too large, and lead to excessive delay and area.

2.2 Generalized Parallel Counter (GPC) Configuration Circuit (GPCCC)

Let $B = b_{s-1}b_{s-2}...b_0$ be an *s*-bit unsigned binary integer, where b_0 is the *least significant bit (LSB)* and b_{s-1} is the *most significant bit (MSB)*. The subscript *r* of bit b_r is called the *rank* of b_r. Each bit of rank *r* contributes a total value of $b_r 2^r$ to the value of *B*.

Given a set of input bits to sum, a *column* is the set of bits of the same rank from each integer. An *m:n* counter takes in *m* input bits from the same column, each having rank *r*, and produces *n* output bits of rank *r*, *r+1*, …, *r+(n-1)* respectively. A *Generalized Parallel Counter (GPC)* [18] is an extension of a parallel counter that can compress input bits from multiple columns. Formally, a GPC is defined as a tuple $(k_{t-2}, k_{t-3}, ..., k_0; n)$, where k_r is the number of input bits of rank *r* summed by the counter, and *n* is the number of output bits.

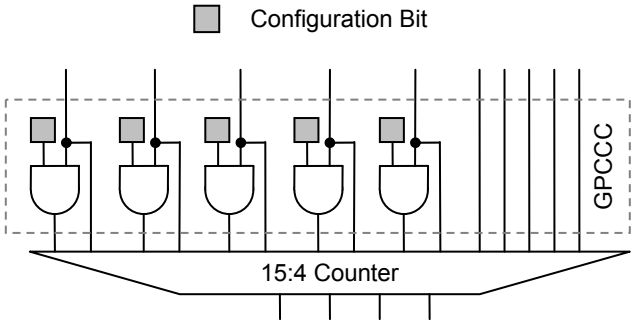

Figure 2.

Illustration of a GPCCC.

For example, a *(5, 5; 4)* GPC sums five bits of rank *1* and four bits of rank *0*; the maximum output value is *15*, so *n=4* output bits are required. Leading zeros are generally omitted from a GPC definition, e.g., one would write *(5, 5; 4)* in lieu of *(0, 5, 5; 4)*.

One implementation of a GPC is to use an *m:n* counter, where each input bit of rank *r* is connected to 2^r GPC inputs. This architecture requires that $m \geq 2^{t-1} + 2^{t-2} + ... + 1$; any unused inputs can be driven to *'0'*.

The *GPC Configuration Circuit (GPCCC)* shown in Fig. 1 extends the largest *m:n* counter in the chain so that it can implement a variety of GPCs. Fig. 2 shows an example in which a GPCCC placed in front of a *15:4* counter allows it to implement a (5, 5; 4) GPC. The GPCCC has ten inputs and fifteen outputs. The five inputs on the right-hand-side of Fig. 2 have rank *0*; the five inputs on the left-hand side can be configured as rank *0* or *1* by setting the appropriate configuration bits. Thus, this specific GPCCC architecture allows the *15:4* counter to be configured as an *m:n* counter with up to *10* inputs, or a variety of different GPCs: *(5, 5; 4)*, *(4, 6; 4)*, *(3, 7; 4)*, *(2, 8; 4)*, and *(1, 9; 4)*.

For each FCS the DSE enumerates all GPCCCs that can extend it without exceeding the input or output requirements. We found the number of GPCCCs to be tractable for the *15:4* and *31:5* counters explored here, but prohibitively large for *63:6* counters.

2.3 Input Configuration Circuit (ICC)

The *Input Configuration Circuit (ICC)* in Fig. 1 allows the *31:5* counter to implement smaller counters, e.g., by setting two input bits to *'0'*, it can implement a *29:5* counter. Given a GPCCC, the ICC is derived deterministically: if the GPCCC has *m* input bits, then an *m*-input, *m*-output ICC is required. A configuration bit is required for each wire, along with an *AND* gate; setting each configuration bit to *'0'* independently drives each ICC output to *'0'*.

2.4 Maximum Output Rank Configuration (MORC)

Each CSlice can be configured to produce more than one output bit by replicating portions of the counter chain after the largest counter. If the chain is replicated *k* times, then the Cslice can be configured to produce *1* to *k* output bits of ranks *0* to *k – 1*; *k – 1* in this case is called the *Maximum Output Rank Configuration (MORC)*. The *Output Rank Configuration (ORC)* is the number of output bits that the CSlice is currently configured to produce.

2.5 Output Multiplexing Circuit (OMC)

The carry bits propagated from one CSlice to the next differ, depending on the ORC. If the MORC is $k - 1$, then a *k:1* multiplexer is required to select among *k* different possibilities for each carry-out bit. The *Output Multiplexing Circuit (OMC)* contains a multiplexer on each carry bit that is propagated from one CSlice to the next: the OMC is not needed if the MORC is *1*. The OMC can be derived deterministically from the FCS and MORC; although it influences both the delay and area of the circuit, it is not an independent parameter that is varied by the DSE.

Increasing the MORC increases the area of a CSlice by replicating the chains of counters and bitwidth of the CPA, allowing each CSlice to produce multiple output bits. Although this yields a larger CSlice, it can reduce the number of CSlices required to map each benchmark. The CSlice area is dominated by the area of the largest counter; thus, increasing the MORC allows the mapper to use fewer CSlices overall, which yields better area utilization. The drawback is that the extra multiplexers in the OMC increase the critical path delay through each CSlice.

2.6 Carry Propagate Adder (CPA) Chain Interrupt Configuration Circuit (CICC)

A *carry-propagate adder (CPA)* [7, Fig. 8] performs the final addition of the *sum* and *carry* outputs of the compressor tree. To support an *ORC* of rank *j*, a *j*-bit CPA is required; the carry-out bit is propagated to the next CSlice. To support a *MORC* of rank *k*, any *k*-bit CPA can produce the sum bits; however, all of the carry-out bits for rank *j*, $1 \leq j \leq k$ are required, since any of these bits could be propagated to the next CSlice via the OMC.

The *Chain Interrupt Configuration Circuit (CICC)*, shown on the right-hand side of Fig. 1, allows the user to program the carry-in bit of each CSlice to *0*. A single multi-input addition or multiplication operation may not use the complete FPCT; interrupting the carry-chain permits a second independent operation to use the remainder of the FPCT. This functionality was not considered during our DSE; nonetheless, the CICC was generated and synthesized for each CSlice for completeness.

3. Design Space Exploration

The following subsections describe the DSE platform that was used in our experiments. The platform, which is shown in Fig. 3, consists of several elements:

- Generic HDL models that hierarchically describe the FPCT.
- Perl scripts that perform the DSE operations: configuring the HDL models, mapping input compressor trees onto the FPCT, invoking the synthesis tool, and extracting timing and area results from reports generated by the synthesis tool.
- TCL scripts that synthesize the FPCT, remove false paths, and generate the required timing and area reports.
- Other tools and scripts used to format the DSE results, etc.

3.1 FPCT HDL Model

The FPCT/CSlice architectures are modeled completely in VHDL. Generic module design capabilities are used to parameterize the FPCT/CSlice architectures. A few parameters, such as the FCS, are calculated offline by the generator script and the corresponding architectural components are written to a VHDL package which is used by the appropriate design modules.

Fig. 3 shows the structure of the model. Solid lines represent hierarchical dependencies and dependencies between modules, while dashed lines represent parameterized dependencies based on the high-level package, *fpct_pkg*. *fpct_top* is the top-level module, *cslice* is the CSlice model, *cntr* models a generic *m:n* counter, and *ICC*, *GPCCC*, *CICC*, and *CPA* respectively model the components of the same name; *sreg* models a shift register (used by the CICC, ICC, OMC, and GPCCC); and, the package *GPCCC_block* contains sub-blocks used to construct the GPCCC.

The model used is generic in terms of the three parameters that characterize the FPCT (the FCS, GPCCC/ICC, and MORC), and was developed using a synthesizable subset of VHDL. Each FPCT sub-block was modeled in a generic fashion. Sub-blocks were connected together to form higher-level blocks, which were also generic. Most of the generic blocks were modeled using generic statements in VHDL; others were calculated using a Perl script and written to a VHDL package included by other modules; the remainder of the model was developed in pure VHDL.

The parallel counters were modeled using a generic compressor tree built from full- and half-adders, which was then optimized using *Synopsys Design Compiler v2006.06* using the *compile_ultra* option. The implementation used a *TSMC 90nm* process with an *Artisan* standard cell library.

The correct propagation of carry-out bits produced by the parallel counters from the current and previous CSlices results in an intricate interconnection scheme of counters, OMCs, CICCs, and CPAs. The interconnection in a top-level CSlice was modeled using a combination of process statements and VHDL functions. This approach yielded correct and fully verifiable VHDL code.

The DSE considers two FCSs, *15:4* and *31:5* and three MORCs, *1*, *2*, and *3*. This results in CSlice architecture descriptions for which only the GPCCC/ICCC is varied. Performing complete synthesis of the architecture for each case significantly increases the exploration time and introduces non-determinism due to the specific algorithms used for optimization by *Synopsys Design Compiler*. Therefore, the non-GPCCC/ICC portions of the six baseline CSlice architectures were synthesized and optimized separately and saved in a library. During the DSE, only the GPCCC/ICC's are generated anew and synthesized; the rest of the CSlice is invoked from the library.

3.2 Exploration Module

The *exploration* scripts (one for each FCS) are the top-level modules called by the user during DSE. The input includes the input bit pattern (benchmark) and the MORC. The exploration script, illustrated in Fig. 4, systematically enumerates each GPCCC/ICCC that can fit the FCS; for each GPCCC/ICC it then performs the following steps:

- The *fpct_gen* script is invoked to generate the FPCT. This script, in turn, invokes the *fpct_pkg* module, which holds some constants that were calculated offline, a testbench based on *System Verilog* (see Subsection 3.3), and a TCL script for proper synthesis of the FPCT.
- The *mapping* script (see Subsection 3.4) is invoked to synthesize the input bit pattern onto the FPCT.

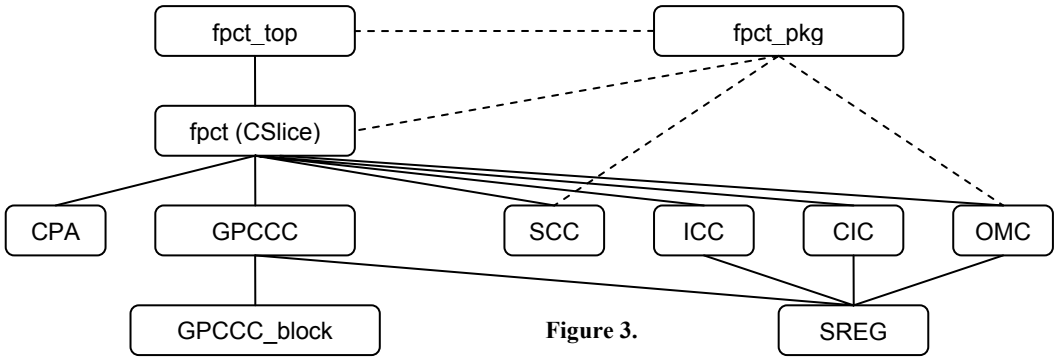

Figure 3.
FPCT HDL Model Structure

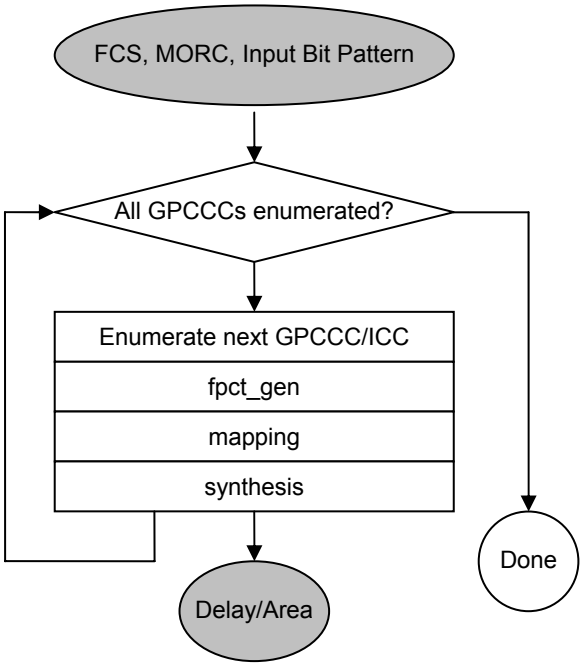

Figure 4.
Exploration module

- *Synopsys Design Compiler* is invoked to synthesize the design and generate delay and area reports.

The *exploration* script is invoked once per benchmark, as the number of CSlices required varies.

3.3 Model Verification

The model was verified using a generic testbench developed in *SystemVerilog*. A white box-based approach was taken to ensure coverage of every corner of the verification space. Let FCS_{out} be the number of output bits of the largest counter in a CSlice. An FPCT comprised of $2FCS_{out} - 1$ CSlices is generated; the middle CSlice is the design under test. This value was chosen so that all possible bit propagations from (to) preceding (subsequent) CSlices will occur in the middle CSlice.

In all preceding CSlices, the ORC is configured to the minimum required for full bit propagation; the *ORC* is configured as the *MORC* for all subsequent CSlices. The ICC for the CSlice under test and all preceding CSlices is set to accept all input bits, but is disabled for subsequent CSlices which are used to propagate the carry-out bits of the CSlice under test. The CICC is enabled in all CSlices, except for the first.

The simulation proceeded as follows. In an outer loop, a random ORC for the CSlice under test is chosen. The FPCT is configured using these values together with the appropriate configurations of the OMC, CICC, and ICC, which are derived deterministically. Within an inner loop several input bit patterns are generated and fed into the FPCT. A summation of the input bit patterns is calculated by the testbench and compared against the output of the FPCT. An error is generated if a mismatch occurs; all errors were debugged prior to running the experiments.

3.4 Mapping Heuristic

The mapping heuristic was written in Perl and embedded in the *mapping* module. Its output is a TCL script used by the synthesis tool. The script is a sequence of *set_case_analysis* commands that replace each configuration flip-flop in the FPCT with a constant. Replacing flip-flops with constant values effectively configures the FPCT, closing many false paths that would otherwise occur; however, it is important that these flip-flops are not treated as constant values that could be optimized via propagation. This ensures that the critical path reported by the timing analyzer corresponds precisely to the critical path of the input bit pattern synthesized on the FPCT.

The mapping heuristic determines the number of CSlices required for each benchmark. Unlike our previous work [7], which used multiplce FPCTs to realize large compressor trees, our script generates a complete FPCT that precisely matches the needs of each benchmark. This eliminates the non-determinism that arises from inter-FPCT routing delays; furthermore, this is highly dependent on the placement of FPCTs within a larger FPGA—which is beyond the scope of this work. Another drawback is the high runtime of FPGA placement and routing, which would significantly impede the DSE. Therefore, if a benchmark cannot be mapped onto the FPCT because the FCS is small or the GPCCC too restrictive, then the design is not evaluated.

The mapping heuristic solves a problem outlined in our previous work, but has been simplified in order to reduce its runtime. It employs a greedy right-to-left pass over the input columns, starting with the least significant column in terms of rank. At each

step, the heuristic generates a new CSlice and attempts to map bits from the current column onto it. The heuristic grabs as many bits as possible from the current and subsequent columns, but is limited by the GPCCC and MORC; it then maps each of these bits onto the current CSlice. If any bits in the current column remain, then the mapping fails—a "vertical configuration" [7, Fig. 9(b)] would be required for this benchmark, indicating that a larger FCS or less restrictive GPCCC would be ideal for this benchmark.

If no bits remain from the current column, then the heuristic sets the ORC of the current CSlice appropriately (e.g., [7, Fig. 6]), and moves on to the next column. After processing all columns, additional CSlices are generated to propagate all of the carry-out bits until the complete sum is produced. This eliminates the need for "horizontal" configurations [7, Fig. 9(a)], where one FPCTs carry-outputs are propagates to the carry-inputs of the next.

4. I/O UTILIZATION: PRUNING THE DSE

Let a *family* be a set of FPCTs with a given FCS and MORC. Within a family, we observed that the delay and area among the FPCTs within each family strongly correlated to the number of CSlices required, which we denote by *N*. Intuitively, reducing the number of CSlices reduces area; however, it also reduces the critical path delay through the CPA. Thus, one goal of the *exploration* module is to find the GPCCC that simultaneously maximizes (1) the number of input bits mapped to each CSlice and (2) the number of output bits produced by each CSlice. This information can be determined from the mapping phase of the *exploration* module shown in Fig. 4; synthesis via *Synopsys Design Compiler*—the most runtime-intensive portion of the DSE—can therefore be eliminated.

The *input utilization* of a CSlice measures its ability to consume input bits. The most obvious measurement of input utilization is the number of input bits mapped to each CSlice; however, this is skewed by the GPCCC. For example, let *FCS = 15:4*; a GPCCC of *(0, 15; 4)* allows up to fifteen inputs; on the other hand, a GPCC of *(5, 5; 4)* has up to ten inputs, but has greater flexibility in terms of mapping. Comparing the input utilization of the two is difficult, since in the end, up to fifteen input bits of the FCS will be used in both cases (see Fig. 2). Suppose that *N* CSlices are used and the FCS is an *m:n* counter; now, let *X* be the total number of input bits that are *not* driven to *'0'* by the ICC after mapping; *X* ignores bits mapped to the final CSlice, since it may be under-utilized due to a lack of available bits rather than poor utilization; *X* also ignores the additional CSlices that are appended to the FPCT to propagate the carry bits.

The input utilization is defined as the quantity $U_{in} = X/Nm$. For example, if our CSlice is configured as a *(5, 5; 4)* GPC and two bits of rank *1* and four bits of rank *0* are mapped onto the CSlice, then $U_{in} = (2\times2^1 + 4\times2^0)/15 = 8/15 = 0.53$.

The *output utilization*, U_{out}, is defined for CSlices whose MORC exceeds *1*. Recall for a given MORC of *k*, the ORC can be configured to any value *j*, $0 \le j \le k-1$, i.e., the CSlice can produce *1* to *k* output bits. Let O_i be *1* plus the ORC of the i^{th} CSlice in the FPCT. Then,

$$U_{out} = \frac{\sum_{i=1}^{N} O_i}{k(N-1)}. \qquad (1)$$

By construction, U_{out} is constant if the MORC is *0*, as each CSlice that is used produces exactly one output bit.

We observed a strong correlation between the input and output utilization of the different FPCTs that were enumerated during the DSE for MORCs of *1* and *2*. Due to this strong correlation, we introduce a unified *I/O utilization* metric, $U = U_{in}U_{out}$ which can be computed for each FPCT that is generated for each benchmark. Only the FPCTs with the highest I/O utilization values are then synthesized during the DSE. Although there is no formal guarantee of optimality in terms of either delay or area, I/O utilization found, a-priori, most of the best FPCT architectures that were enumerated for each benchmark in our experiments.

5. EXPERIMENTAL RESULTS

5.1 Benchmarks

We selected a set of seven arithmetic benchmarks to use in the DSE; our goal was to find a mix of benchmarks that had a wide variety of bit patterns (e.g., rectangular for multi-input addition, trapezoidal for multiplication, irregular for filters, etc.). In principle, these experiments could easily be repeated with a larger set of benchmarks and fewer restrictions on the FPCT configuration (e.g., consider a wider variety of FCSs).

Table 1 lists the benchmarks, which include compressor trees for three different multipliers, two multiinput addition operations, a FIR filter [7, Fig. 5], and the *Sum-of-Absolute Difference (SAD)* computation, which is used for *motion estimation* in *video coding* algorithms such as *H.264/AVC* [8]. *mul5x5* was selected based on an anecdote in a paper by Kuon and Rose [12]: *mul5x5* performs better on the general logic of an FPGA than on the dedicated *9x9* multiplier in the embedded DSP blocks. *mul36x18* represents either *36x18*-bit multiplication or *18x18*-bit multiplication with Booth encoding. *mul18x18*, *mul36x18*, *add16+16*, and *FIR* were too large to fit on an FPCT whose CSlices have an FCS of *15:4*; the remaining benchmarks fit on FPCTs whose CSlices have an FCS of either *15:4* or *31:5*.

5.2 DSE Results

This section summarizes the results of the complete DSE. For each FCS size (*15:4*, *31:5*), the DSE enumerated every legal GPCCC/ICC and MORC combination, generated and synthesized each FPCT, and then mapped each benchmark as described in Section 3, yielding delay and area measurements. For each FPCT architecture enumerated during the exploration, the delay and area are averaged across the set of benchmarks.

Fig. 5 shows the area/delay results for the 3 benchmarks that could be mapped onto FPCTs whose CSlices have FCS = *15:4*; due to the small number of GPCCC/ICCC and MORC configurations, Fig. 5 reports results for every FPCT.

Table 1.

Benchmark circuits used for the FPCT DSE.

Benchmark	Description	FCSs Mapped
mul5x5	5x5 Multiplication	15:4, 31:5
mul18x18	18x18 Multiplication	31:5
mul36x18	36x18 Multiplication	31:5
add8x32	Add 8 32-bit Integers	15:4, 31:5
add16x16	Add 16 16-bit Integers	31:5
FIR	FIR Filter	31:5
SAD	Sum-of-Absolute-Differences	15:4, 31:5

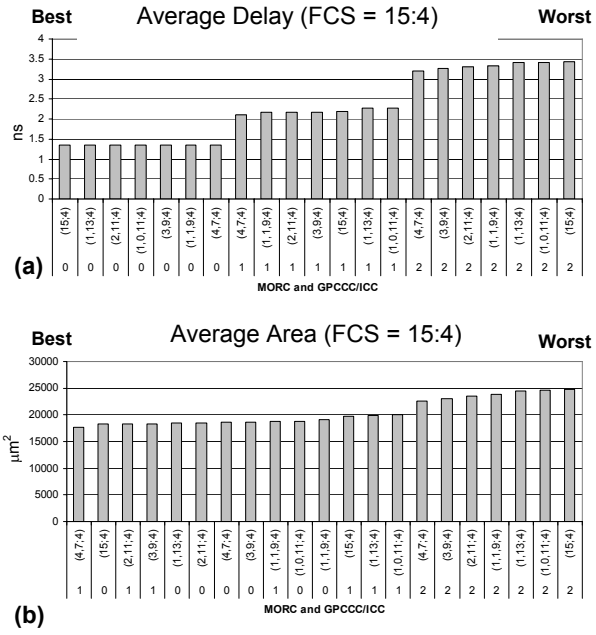

Figure 5.

Average delay (a) and area (b) of each FPCT architecture with FCS = 15:4; three of seven benchmarks were used.

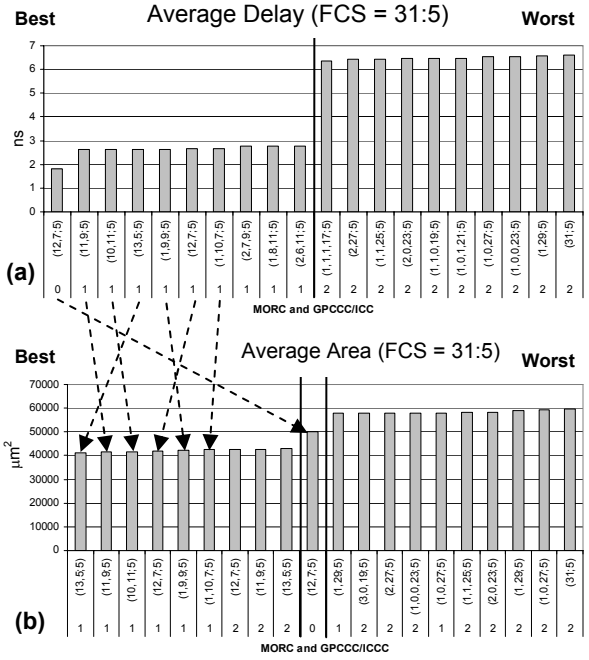

Figure 6.

Average delay (a) and area (b) of the ten best and worst FPCT architectures with FCS = 31:5; all benchmarks were used. The dashed lines link the seven best architectures in terms of delay with their respective areas.

It is important to note that Fig. 5 only contains data points for three benchmarks, as the others were too large to fit onto a single FPCT whose CSlices have FCS = *15:4*. Therefore, we conclude that a larger FCS will be preferable. Nonetheless, Fig. 5 does illustrate several interesting trends.

The average delay reported in Fig. 5(a) correlates strongly with the MORC. The delays are clustered around 3 data points: all FPCTs with a MORC of *0* had delays of approximately *1.4ns*; all FPCTs with a MORC of *1* had delays ranging from approximately *2.1* to *2.3ns*; and all FPCTs with MORC of *2* had delays ranging from approximately *3.2* to *3.4ns*. It appears that for this portion of the design space, the delays through the multiplexers of the OMC distinguish the 3 data points.

The average areas reported in Fig. 5(b) range from approximately *17,500* to *25,000*μm^2. The worst data points in terms of area were those with a MORC of *2*. Among the FPCTs with a MORC of *0*, the smallest tended to be those with few rank-1 and rank-2 input bits. Among the FPCTs with a MORC of *1*, the best tended to be those with a larger number of rank-1 inputs; the design points having both rank-1 and rank-2 inputs in addition to rank-0 inputs tended to be among the larger design points.

Fig. 6 shows similar results for the FPCTs with FCS = *31:5*; here, the total number of FPCT architectures in the design space is too large to enumerate. Therefore, Fig. 6(a) shows the ten best and ten worst architectures in terms of area; Fig. 6(b) shows the nine best and ten worst architectures in terms of area along with the area of the architecture from Fig. 6(a) that has the best delay. Dashed arrows link the seven best architectures in Fig. 6(a) in terms of their delay with their areas.

In Fig. 6(a), the best FPCT architecture in terms of delay has a GPCCC/ICC of *(12, 7; 5)* and a MORC of *0*; the next nine best architectures all have MORCs of *1*; the ten worst architectures all have MORCs of *2*. The delay of the best FPCT architecture is approximately *1.8ns*, while the delay of the second through ninth best range from *2.6* to *2.8ns*. The worst architectures have delays that are between three and four times larger than the best; this illustrates that there is a significant difference in quality between different FPCT architectures, justifying the DSE: an arbitrarily-selected FPCT is not likely to perform particularly well.

Fig. 6(b) shows that the nine smallest FPCT architectures have approximately the same area, ranging from around *41,000* to *43,000*μm^2; the worst FPCT, in contrast, has an area of approximately *60,000*μm^2. Once again, this justifies the DSE on the grounds that an arbitrarily selected FPCT architecture may be among the largest.

Dashed arrows from Fig. 6(a) to (b) associate the best seven best performing FPCT architectures in terms of delay with their respective areas. The fastest FPCT has an area of around *50,000*μm^2, which is somewhere in the middle, in terms of area. The next second through seventh fastest FPCTs are the first through sixth smallest. In terms of Pareto optimality, this yields two points: one point that is optimal in terms of delay (with a significant area overhead) and six approximately equivalent points that are optimal in terms of area and close-to optimal in terms of delay. Depending on the relative importance of delay and area, the designer is free to choose either point.

The fastest FPCT architecture in Fig. 6(a) has a MORC of *0*, and therefore, no OMC; thus, it achieves its speed by eliminating multiplexers in the critical path of the CPA and carry propagation chains from the first CSlice. Since the second fastest FPCT has a MORC of *1*, it contains an OMC, and thus a multiplexer's worth of delay is accumulated through each CSlice; this explains the gap of almost *8ns* between the best and worst.

Because the fastest FPCT architecture has a MORC of *0*, the CPA in each CSlice produces exactly one output bit; consequently, it will require more CSlices than architectures with larger MORCs. The area of the larger counter replicated across more CSlices outweighs the cost of replicating the chain of smaller counters across every CSlice when the MORC exceeds *1*.

5.3 I/O Utilization

As mentioned in Section 4, we observed a strong correlation between the U_{in} and U_{out} values of the different FPCTs that were enumerated during the DSE for MORCs of *1* and *2*; the utilization is constant if the MORC is *0*.

As a representative example, Fig. 7(a) shows U_{in} and U_{out} for the *mul36x18* benchmark for every GPCCC/ICC configuration enumerated for FCS = *31:5*. Due to the size of the figure, the exact GPCCC/ICCs could not be labeled; it suffices to note that *52* different GPCCC/ICCs were enumerated.

Six curves are shown: U_{in} and U_{out} for MORCs of *0*, *1*, and *2*. The correlation is clear from the figure: the design points with high/low U_{in} values tend to have high/low U_{out} values, across all MORCs; this justifies the use of the unified I/O utilization metric $U = U_{in}U_{out}$. In general, the most profitable GPCCC/ICCs to explore tend to be those with the highest I/O utilization.

mul5x5, whose U_{in} and U_{out} values for different GPCCC/ICCs are shown in Fig. 7(b), is an outlier: due to its small size, many GPCCC/ICC configurations achieved similar U_{in} and U_{out} values (for their respective MORCs). Our goal is to use I/O utilization to prune the search space by considering only the FPCT designs for each benchmark whose utilization is maximal. Due to the large number of points with maximal I/O utilization, the search space for *mul5x5* cannot be pruned efficiently with this method.

The number of design points in Fig. 7(a) and (b) differ; *mul5x5* is sufficiently small that it can fit onto any FPCT having FCS = *31:5*, regardless of GPCCC/ICC; *mul36x18*, in contrast, could not fit onto every design that was enumerated; these designs were discarded and their I/O utilization was not reported.

5.4 Pruning the Design Space

Here, we evaluate the effectiveness of pruning the design space with I/O utilization for MORCs of *1* and *2*. Fig. 8 shows all of the points in the design space enumerated for each benchmark for FPCTs with FCS = *31:5*. For MORCs of *1* and *2*, the four points having maximum I/O utilization are circled and labeled (*A-D* for MORC = *1*; *E-G* for MORC = *2*). In all cases except for *add8x32*, the Pareto-optimal points for each MORC are contained within the four points; for *add8x32*, the points that are found are near-Pareto-optimal. For the other benchmarks, the four points per MORC were typically the best; however, there are some exceptions: for example, there are several design points in Fig. 8(d) (*SAD*) that have a lower area and approximately the same delay as points *C* and *D*.

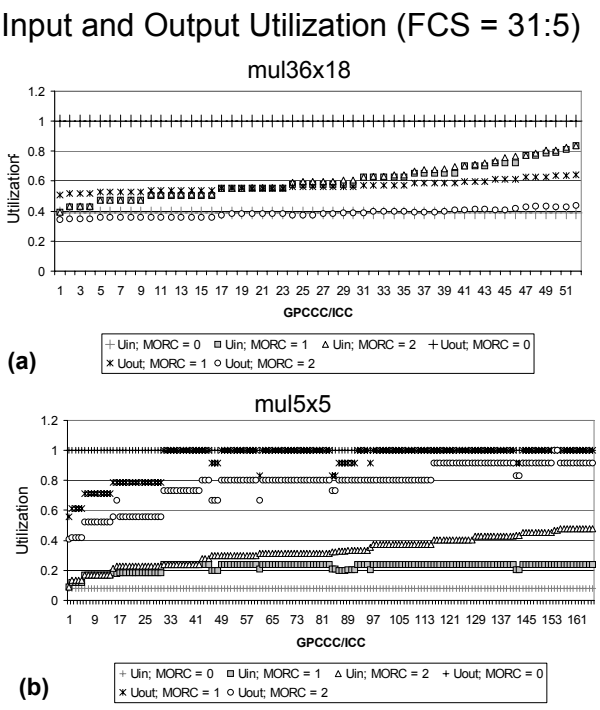

Figure 7.

A strong correlation between input and output utilization was observed; (a) mul36x18 is a representative example; (b) one anomaly is mul5x5, where maximal utilization was observed for many GPCCC/ICC combinations.

Fig. 8 demonstrates that I/O utilization can find near-Pareto optimal points in the design space without a full-blown DSE; however, it may miss some points that are still good solutions. In Fig. 8, the choice of four maximum I/O utilization points per MORC was arbitrary; increasing the number of maximum I/O utilization points per MORC would increase the likelihood of finding Pareto-optimal solutions, but increases the runtime.

It is important to note that I/O utilization requires exhaustive enumeration of the different points in the design space; the pruning criterion reduces the number of designs to synthesize. It is also worth noting that this only works for MORCs greater than zero; when the MORC is *0*, I/O utilization is constant, so all points are equivalent. Other methods may need to be developed for pruning when the MORC is *0*. The effectiveness of pruning, however, suggests that there may exist analytical methods to evaluate different FPCTs without resorting to a pruned DSE; developing such methods is one potential avenue for future research on this topic.

6. RELATED WORK

DSE has been used for many academic and industrial studies for FPGA architecture evaluation. A typical approach, similar to what is done here, is to enumerate a set of different FPGA architectures, place-and-route a set of benchmark circuits, and extract appropriate metrics (e.g., delay, throughput, wirelength, LUT usage, etc.) to evaluate the quality of the architectures under consideration, using tools such as *VPR* [4, 5].

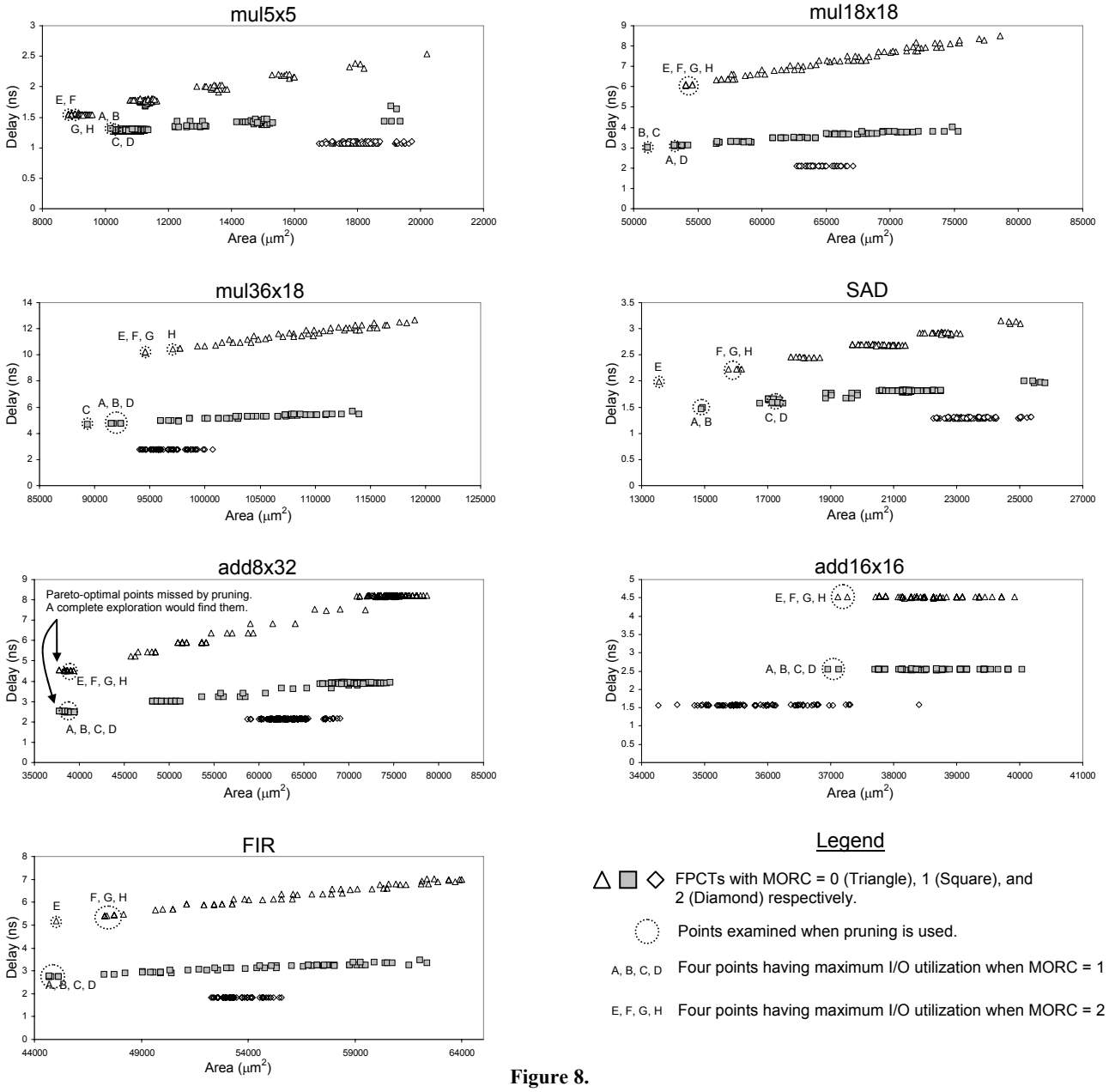

Figure 8.

For each benchmark with FCS = 31:5, the points in the design space found by pruning. The four points whose utilization is maximal when the MORC is 1 and 2, respectively, are shown; utilization is flat if the MORC is 0, so pruning is impossible. In all cases except add8x32 (e), pruning found the Pareto-optimal points when the MORC is 1 or 2.

For example, many parameters of *Altera's Stratix II* logic and routing architecture were determined via DSE [13]; other DSE-based studies include: Ahmed and Rose's experiments on LUT and cluster size [1]; Kuon and Rose's more recent experiments on the effects of varying architectureal parameters (e.g., LUT size) and transistor-sizing vis-à-vis delay and area [11]; and Ye and Rose's evaluation of the use of bus-based interconnections in an FPGA routing network [23].

Reconfigurable arithmetic accelerators are also related to this work; typically, they have been proposed as customizable accelerators for application-specific processors. Chimaera [10], for example, is a LUT-based accelerator with fast carry chains; many of these ideas have since been incorporated into the logic blocks of high-performance FPGAs [13]. Yehia et al. [24] performed a DSE for an accelerator for superscalar processors that can collapse sequential logic chains into a single cycle; the design space included parallel-prefix addition, LUTs to perform

arbitrary logic functions at the bit-level, and optional shifters placed at the inputs and output of the device. Ansaloni et al. [3] recently introduced the *Expression-Grained Reconfigurable Array (EGRA)*, which contains several levels of arithmetic and logic operations connected by programmable switches; the operations supported include logical operations (e.g., *AND*, *XOR*), arithmetic and logical shifts, and addition/subtraction and comparison operations.

The FPCT differs from the preceding arithmetic accelerators in two respects: (1) it is intended for integration into an FPGA, rather than a processor, and (2) it accelerates multi-input addition and multiplication operations, rather than chaining a CPA with other logical operations.

7. CONCLUSION

This paper has presented a DSE methodology that can optimize an FPCT for a given set of benchmarks; the I/O utilization metric was introduced to reduce the number of FPCT architectures that are synthesized during the DSE, while providing high confidence to the user that the remaining design points are among the best. We used a MORC of *2* for our previous FPCT evaluation; as a result of this study, we have observed that a MORC of *1* tends to be preferable in terms of delay and area—at least for the benchmarks examined here. An FPGA vendor who wishes to integrate an FPCT into a large reconfigurable device—such as an FPGA—could use our approach to determine the best FPCT architecture for *their* most important customer's benchmark circuits.

REFERENCES

[1] Ahmed, E., and Rose, J. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. VLSI, vol. 12*, no. 3, March, 2004, 288-298.

[2] Altera Corporation. Stratix II vs. Virtex-4 Performance Comparison. Available online: http://www.altera.com/

[3] Ansaloni, G., Bonzini, P., and Pozzi, L. Design and architectural exploration of expression-grained reconfigurable arrays, *IEEE Symposium on Application-Specific Processors*, Anaheim, CA, USA, June 8-9, 2008.

[4] Betz, V., and Rose, J. VPR: a new packing, placement and routing tool for FPGA research, *7th Int. Workshop on Field-Prog. Logic and Applications*, London, UK, September 1-3, 1997, 213-222.

[5] Betz, V., Rose, J., and Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*, Springer, 1999.

[6] Brisk, P., Verma, A. K., Ienne, P., and Parandeh-Afshar, H. Enhancing FPGA performance for arithmetic circuits. *Design Automation Conf.*, San Diego, CA, USA, June 4-8, 2007, 334-337.

[7] Cevrero, A., et al.. Architectural improvements for field programmable counter arrays: enabling efficient synthesis of fast compressor trees on FPGAs. *Int. Symp. FPGAs*, Monterey, CA, USA, February 24-26, 2008, 181-190.

[8] Chen, C-Y., et al.. Analysis and architecture design of variable block-size motion estimation for H.264/AVC, *IEEE Trans. Circuits and Systems-I, vol. 53, no. 2*, February, 2006, 578-593.

[9] Dadda, L., Some schemes for parallel multipliers, *Alta Frequenza, vol. 34*, May, 1965, 349-356.

[10] Hauck, S., Fry, T. W., Hosler, M. M., and Kao, J. P. The Chimaera reconfigurable functional unit. *IEEE Trans. VLSI, vol. 12*, no. 2, February, 2005, 206-217.

[11] Kuon, I., and Rose, J. Area and delay tradeoffs in the circuit design of FPGAs. *Int. Symp. FPGAs*, Monterey, CA, USA, February 24-26, 2008, 149-158.

[12] Kuon, I., and Rose, J. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design, vol. 26*, no. 2, February, 2007, 203-215.

[13] Lewis, D. M., et al. The Stratix II logic and routing architecture. *Int. Symp. FPGAs*, Monterey, CA, USA, February 20-22, 2005, 14-20.

[14] Mirzaei, S., Hosangadi, A., and Kastner, R. High speed FIR filter implementation using add and shift method, *Int. Conf. Computer Design*, San Jose, CA, USA, October 1-4, 2006.

[15] Parandeh-Afshar, H., Brisk, P., and Ienne, P. Efficient synthesis of compressor trees on FPGAs. *Asia-Pacific Design Automation Conf.*, Seoul, Korea, January 21-24, 2008, 138-143.

[16] Parandeh-Afshar, H., Brisk, P., and Ienne, P. Improving synthesis of compressor trees on FPGAs via integer linear programming. *Design Automation and Test in Europe*, Munich, Germany, March 10-14, 2008, 1256-1261.

[17] Sriram, S., Brown, K., Defosseux, R., Moerman, F., Paviot, O., Sundararajan, V., and Gatherer, A. A 64 channel programmable receiver chip for 3G wireless infrastructure, *IEEE Custom Integrated Circuits Conf.*, San Jose, CA, USA, September 18-21, 2005, 59-62.

[18] Stenzel, W. J., Kubitz, W. J., and Garcia, G. H. A compact high-speed parallel multiplication scheme, *IEEE Trans. Computers, vol. C-26, no. 10,* October, 1977 948-957.

[19] Verma, A. K., and Ienne, P. Automatic synthesis of compressor trees: reevaluating large counters, *Design Automation and Test in Europe (DATE '07)* (Nice, France, April 16-20, 2007) 443-448.

[20] Verma, A. K., and Ienne, P. Improved use of the carry-save representation for the synthesis of complex arithmetic circuits, *Int. Conf. Computer-Aided Design*, San Jose, CA, USA, November 7-11, 2004, 791-798.

[21] Wallace, C. S. A suggestion for a fast multiplier, *IEEE Trans. Elec. Computers, vol. 13*, February, 1964, 14-17.

[22] Xilinx Corporation. Virtex-5 user guide. Available online: http://www.xilinx.com/

[23] Ye, A. G., and Rose, J. Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits. *IEEE Trans. VLSI, vol. 14*, no. 5, May, 2006, 462-473.

[24] Yehia, S., Clark, N., Mahlke, S. A., and Flautner, K. Exploring the design space of LUT-based transparent accelerators. *Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems*, San Francisco, CA, USA, September 24-27, 2005, 11-21.