Memory Organization and Data Layout for Instruction Set Extensions with Architecturally Visible Storage

Panagiotis Athanasopoulos[1] Philip Brisk[2] Yusuf Leblebici[3] Paolo Ienne[1]

[1]School of Computer and Communication Sciences
[3]School of Engineering
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
{First_name.Last_name}@epfl.ch

[2]Department of Computer Science and Engineering
Bourns College of Engineering
University of California, Riverside
Riverside, CA 92521
First_name.Last_name@gmail.com

ABSTRACT

Present application specific embedded systems tend to choose instruction set extensions (ISEs) based on limitations imposed by the available data bandwidth to custom functional units (CFUs). Adoption of the optimal ISE for an application would, in many cases, impose formidable cost increase in order to achieve the required data bandwidth. In this paper we propose a novel methodology for laying out data in memories, generating high-bandwidth memory systems by making use of existing low-bandwidth low-cost ones and designing custom functional units all with the desirable data bandwidth for only a fraction of the additional cost required by traditional techniques.

Categories and Subject Descriptors

B.2.4 [**Arithmetic and Logic Structures**]: Algorithms, Cost/performance; B.3.2 [**Memory Structures**]: Design styles; B.5.2 [**Register-Transfer-Level Implementation**]: Design, Arithmetic and Logic units, Control design, Data-path design, Memory design

General Terms

Algorithms, Performance, Design.

Keywords

Extensible Processor, Instruction Set Extensions (ISEs), Architecturally Visible Storage (AVS).

1. INTRODUCTION

Customizable and extensible processors have become increasingly popular in recent years for deployment in cost-constrained embedded systems [1-5, 9, 10, 12, 13]. The use of a pre-existing base processor core simplifies the design and verification effort; system designers, meanwhile, benefit from the use of well-understood compiler algorithms to identify and synthesize application-specific custom instruction set extensions (ISEs). ISEs typically occur in the most computationally intensive loops in the application. Several recent papers have shown that significant benefits can be reaped from the inclusion of Architecturally Visible Storage (AVS), e.g., local memories, in ISEs [2, 5].

ICCAD'09, November 2–5, 2009, San Jose, California, USA.

This paper focuses on effective synthesis techniques for AVS-enhanced ISEs. The input is an AVS-enhanced ISE that accesses an array, which is stored in AVS. Firstly, we determine a low-cost memory structure that can provide adequate bandwidth to the ISE; the goal is to reduce the cost of the memory without compromising performance. Secondly, we determine a layout of data in the memory structure identified in the first step; the goal is to minimize the total number of load and store operations required to execute the ISE. Lastly, we insert a data alignment layer into the ISE itself; this layer, which is derived from the layout, ensures that all necessary connections exist between the memory input/output ports and each input/output operation in the ISE logic. Altogether, our system ensures a low-cost memory architecture that ensures that the AVS-enhanced ISE achieves optimal performance.

2. Motivating Example

This section uses a 2-dimensional discrete cosine transform (2D-DCT) as a motivating example. The 2D-DCT algorithm used in applications such as JPEG and MPEG operates on an 8x8 matrix of integers. The 2D-DCT repeatedly performs a 1-dimension DCT (1D-DCT) on the rows and columns of the matrix. It suffices to note that the 1D-DCT is an 8-input, 8-output computation whose outputs overwrite the inputs in the context of the 2D-DCT. In short, the 2D-DCT proceeds by computing a 1D-DCT over each row of the matrix, followed by a 1D-DCT over each column. As each 1D-DCT overwrites its inputs, the 1D-DCT must be computed for the rows before the columns due to data dependencies.

For simplicity, assume that the 1D-DCT logic requires more than one cycle to compute its result. Consequently, the performance optimal memory structure would provide 8 read ports and 8 write ports; however, such a memory would be far too costly to implement in full-custom SRAM technology. For example, the memory compiler used in our laboratory provides memories with at most 2 I/O ports. A more cost-effective memory organization would be to use 8 smaller memories, each having 1 read and 1 write port. As shown in Fig. 1, this organization is perfect for the row access pattern, but places all of the data in a column in the same memory. Fortunately, the data can be laid out in memory such that one element from each column, in addition to each row, is placed in a single memory; thus, both the rows and columns can be accessed in parallel.

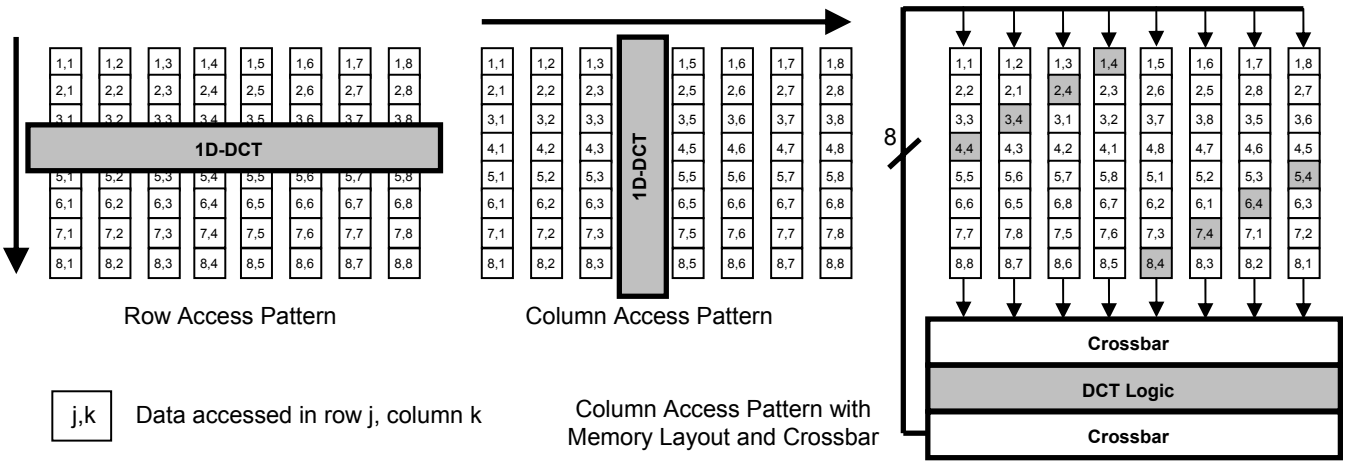

Figure 1.

When 8 cost-effective (1 read port, 1 write port) cost effective memories are used, the column accesses all go to the same memory, which implicitly serializes the computation. This problem can be eliminated by laying out the data in memory so that both the rows and columns can be accessed in parallel. The crossbar is necessary because the DCT is non-commutative.

If the DCT was commutative, this memory layout would suffice; unfortunately, it is not. This is not a problem for the row access pattern, as inputs 1..8 are always read from memories 1..8 respectively; however, this is problematic for the column access pattern: for each memory, there is a column access that requires the data that is read to be sent to each input of the DCT logic. To facilitate these connections, we place a crossbar on the output of the 8 single read/write port memories; this permits the output of every memory to connect to every input of the DCT logic; a similar crossbar will be needed on the output as well, as each DCT output must be able to write to each memory.

To summarize, a memory organization that provides 8 read ports and 8 write ports is needed to ensure optimal performance; however, this cannot be achieved at a reasonable cost with a single multi-ported memory. A lower-cost organization uses 8 individual memories, each of which contains 8 entries and has one read port and one write port. To facilitate single-cycle read and write accesses, the data must be laid out in memory in a specific way, and, since the DCT is non-commutative, a crossbar must be placed between the memory read ports and the DCT logic.

The following section describes, algorithmically, how we determined the solution described in this section.

3. OPTIMIZING THE MEMORY SYSTEM FOR AVS-ENHANCED ISES

We assume that AVS-enhanced ISE identification has been performed in advance; the specific method used is not a primary concern. The information required to optimize the memory system is which arrays will be stored in the local memory and the desired I/O bandwidth; this information can be determined from each ISE and/or the compiler that identifies it. Our synthesis flow will generate memories from an existing vendor's tools; we only consider memories that this tool can generate. Consequently, our exploration is limited by vendor specifications, such as the minimum/maximum number of I/O ports, word capacity, etc.

Given this information, our algorithm to optimize the memory system for an AVS-enhanced ISE proceeds as follows.

Enumerate Memories. We generate all valid SRAM memories that meet the vendor's specifications, as well as flip-flop memories when the capacity is sufficiently small.

Memory Organization. We exhaustively consider all possible combinations of memories that have sufficient capacity and can provide the desired bandwidth for a performance-optimal implementation of the ISE. An exhaustive enumeration is permissible in this context because the number of memories enumerated in the preceding step is quite small, and the number of unique valid combinations of them is also limited by the desired bandwidth of the ISE.

Cost Estimation. The cost of each memory organization that meets the specifications is evaluated. The cost can be estimated using area numbers provided by the memory vendors, or coarse-grained estimates in terms of gate and multiplexing costs. The different organizations are then sorted by cost.

Data Layout. A memory organization is feasible if there is at least one layout of data stored in the memory that can guarantee that all operands will be available to the ISE in a single cycle. The minimum cost feasible memory organization will be selected for inclusion in the ISE. The process uses a linear search, starting from the cheapest memory organization; the first organization for which a feasible layout is found is selected.

Alignment Layer. The cheapest feasible memory organization, after a layout is found, may not provide the requested operands in the correct order for each execution of the ISE, as in the DCT example in the preceding section. This occurs because the data layout algorithm simply computes a feasible alignment, but without considering the order of the operands produced by the memory. Our solution is to include an alignment layer, e.g., a crossbar, between the memory read ports and the ISE logic. The alignment layer can be (partially) eliminated if the ISE is (partially) commutative.

4. DATA LAYOUT OPTIMIZATION

This section presents an algorithm to compute a feasible data layout for a given memory system. The algorithm, Limited Improper Constrained Color Assignment (LICCA), generalizes the well-known k-Colorability problem and is NP-complete.

4.1 LICCA Problem Formulation

First, we formalize the LICCA problem in the abstract, following which we relate it to the memory layout problem at hand.

The input to LICCA is a graph G = (V, E, I), having sets of vertices $V = \{v_0, \ldots, v_{n-1}\}$ and edges $E = \{e_0, \ldots, e_{m-1}\}$, along with a set of sets of vertices $I = \{I_0, \ldots, I_{L-1}\}$ such that $E = \{(v_x, v_y) | \exists I_j \in E \ni v_x \in I_j \text{ and } v_y \in I_j\}$; in other words, an edge (v_x, v_y) exists in E if and only if v_x and v_y both belong to at least one set I_j. Additionally, we are given a positive integer k, and two sets of positive integers $N = \{n_0, \ldots, n_{k-1}\}$ and $A = \{a_0, \ldots, a_{k-1}\}$.

A solution to LICCA is an assignment of colors (integers) to vertices, i.e., a function $f: V \rightarrow \{0, \ldots, k-1\}$. A legal solution satisfies the following properties:

(1) A maximum of n_i vertices can receive color i, $0 \leq i \leq k-1$; that is, $|\{v \in V | f(v) = i\}| \leq n_i$.

(2) For each set $I_j \in I$, there can be at most a_i vertices that receive color i.

Any instance of the k-colorability problem can be reduced to an instance of LICCA by setting $I = \{\{v_x, v_y | (v_x, v_y) \in E\}\}$, and, for $0 \leq i \leq k-1$: $n_i = |V|$ and $a_i = 1$. Intuitively, the set of sets I indicates that some variant of k-colorability must be tested on multiple induced subgraphs of G, rather than on G as a whole; N imposes an upper bound on the number of vertices that can receive a color; and the set A permits an allowable number of coloring constraint violations on a per-induced subgraph basis.

4.2 Discussion and Example

To relate LICCA back to the problem of data layout in memory, V is a set of array elements that must be stored in the AVS of an ISE. An edge $e = (v_x, v_y)$ indicates that v_x and v_y are read in the same cycle, and therefore must reside in distinct memories. Each set of vertices $I_j \in I$ is a set of vertices that are read in parallel by a call to an ISE. *k* is the number of memories in the organization for which we are searching for a layout. n_i is the capacity of the i^{th} memory; and a_i is the number read/write ports of the i^{th} memory.

Fig.; 2 shows an example that illustrates LICCA. Assume that the array in question has 6 elements $V = \{v_0, v_1, v_2, v_3, v_4, v_5\}$, that the ISE reads/writes 3 values from the array, and that we can statically determine that across all invocations of the ISE, it will read three distinct combinations of array elements: $I_0 = \{v_0, v_1, v_2\}$, $I_1 = \{v_3, v_4, v_5\}$, and $I_2 = \{v_0, v_2, v_5\}$. This implies that the resulting graph contains 8 edges, i.e., $E = \{(v_0, v_1), (v_0, v_2), (v_0, v_5), (v_1, v_2), (v_2, v_5), (v_3, v_4), (v_3, v_5), (v_4, v_5)\}$.

Now, assume that there are two memories, M_0 and M_1. M_0 has a capacity of 4 ($n_0 = 4$) and two ports ($a_0 = 2$), while M_1 has a capacity of 2 ($n_1 = 2$) and one port ($a_1 = 1$). The fact that there are two memories implies that k, the number of available colors, is two. In this case, a valid color assignment exists. Vertices v_0 and v_4 are assigned color 1, while the remaining vertices are assigned color 0; this assignment satisfies the capacity constraints of M_0 and M_1.

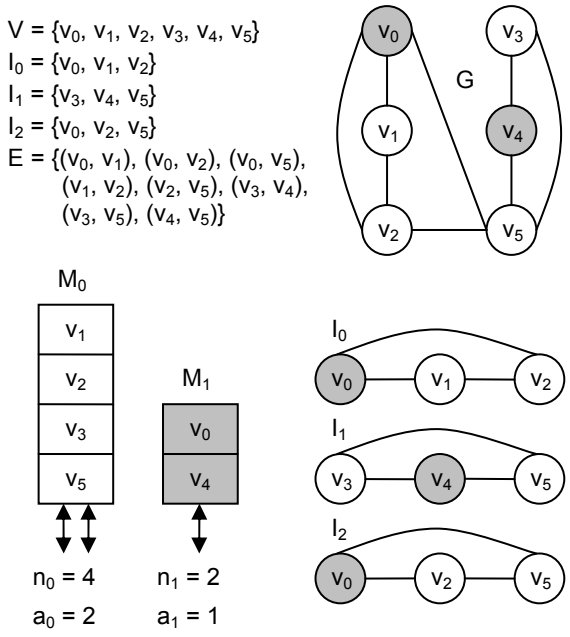

Figure 2.

An example of LICCA. The color assignment shown is not a legal k-coloring of G; however, it is a legal solution to LICCA, because at most two vertices in each set I_0, I_1, and I_2 receive color 0 (white). This is sufficient because the corresponding memory (M_0) has two I/O ports.

Without loss of generality, consider $I_0 = \{v_0, v_1, v_2\}$; v_1 and v_2 are both assigned color 0; since $a_0 = 2$, this coloring violation can be tolerated, since v_1 and v_2 are both assigned to M_0, which has two ports. An analogous constraint is satisfied for I_1 and I_2 as well.

Next, we consider the 2D-DCT from Fig. 1. The array contains 64 elements, and there are 16 accesses, each to 8 elements (8 row accesses, and 8 column accesses), i.e., $|I| = 16$. There are 8 memories ($k = 8$), each with a capacity of 8 ($n_i = 8$, $0 \leq i \leq 7$) and a single port ($a_i = 0$, $0 \leq i \leq 7$). In this case, LICCA has many equivalent solutions; one is shown on the right side of Fig. 1.

4.3 LICCA Implementation

An exact solution to LICCA is required to produce a legal layout. We have implemented an optimal backtracking search that solves LICCA in exponential time in the worst case. For our 2D DCT/FFT experiments, LICCA requires less than one minute to converge to a solution. As our implementation proved to be sufficient for layout searches involving up to 1000 elements (as is the case in our benchmarks), we did not attempt to optimize it for speed, for example, by introducing pruning criteria or by upgrading it to an A* Search; also, we did not use heuristics that cannot guarantee to find a legal solution if one exists. In cases where the exact search cannot converge in a reasonable amount of time, one can rapidly retrieve legal solutions based on the observation that access patterns usually lead to observable patterns in the layout. By performing LICCA on a reduced problem size, a solution can be found that can generalize to the larger problem instances. For example, using the 8x8 layout shown in Fig. 1, we generalized the legal solution to a much larger DCT of dimensions 4096x4096, which has more than 16 million elements.

5. ALIGNMENT LAYERS AND DECODERS

Consider port j of memory M_i. Across all memory accesses, there is no guarantee that the data read from this port should always be directed to the same input port of the IE logic. In the most general case, each read/write port of each memory will connect to each input/output port of the ISE logic. The alignment layer refers to the circuitry that facilitates these connections. In the most general case, the alignment layer can be realized as a crossbar.

As part of the ISE control logic, we employ a custom decoder to translate the access id to generate the correct control signals to the alignment layer. In the case of the 2D-DCT example from Section 2, the memory organization has 8 single-port memories, and the ISE logic has eight inputs and outputs. The input and output alignment layers will contain up to eight 8-input multiplexers, each of which will require 3 control signals; hence, the decoder must generate up to 24 control signals for each alignment layer.

For the sake of simplicity, let us assume that no two inputs of the 1D-DCT function are commutative. Using Fig. 1 as a reference, we will determine the input/output alignment layers. Given a memory system organization and a data layout, the process of generating the alignment layer is mostly deterministic; however, the choice of encoding schemes can affect the controller design.

Firstly, observe from Fig. 1 that there are 16 unique accesses: 8 row accesses and 8 column accesses. Consequently, 4 bits are required to encode these accesses, so we do not need to support every possible alignment of data. Looking at the column access patterns, each memory port must connect to each input of the DCT logic and each DCT logic output must connect to each memory port as well. The connections from memory read ports to DCT logic input ports are shown in Table 1. The controller can be derived directly from this table; numerous algorithms exist to optimize the decoder design. The alignment layer between the DCT logic outputs and memory write ports is similar in principle.

6. EXPERIMENTAL RESULTS

This section evaluates the effectiveness of our techniques to reduce the area of AVS-enhanced ISEs. We demonstrate that the memory organization and data layout found by our tool reduces the cost, compared to a multi-ported memory, without sacrificing performance. We also compare our results to more cost-effective memory organizations that provide less I/O bandwidth than ours. All of our experiments use a 0.18μm UMC standard cell library, and a memory compiler targeting the same technology.

6.1 Benchmarks

We considered two benchmarks which we use as case studies: an 8x8 2D-DCT and an 8x8 2D-FFT. The ISE that performs the 2D-DCT the row-column implementation described in Section 2, while the 2D-FFT is a twelve-butterfly unit implementation. The benchmarks are representative of a variety of 2D convolution algorithms that can be decomposed into successive applications of a 1D algorithm applied to the rows and columns of a 2D matrix. Similar algorithms include Finite Impulse Response (FIR) filters, Gaussian Blur filters, Sobel filters, and matrix multiplication.

6.2 Memories Evaluated

For our experiments, we considered four different memory organizations. Both of our benchmarks require memories with 64 entries. The four memory organizations are as follows:

8x8 Non-clustered RF: One 64-entry register file with 8 read ports and 8 write ports; with 8 read and 8 write ports, the ISE logic does not require an alignment layer. This organization suffices for 64 entries, but would not scale in the general case.

8x8 Clustered RF: Eight 8-entry register files, each with a single read and write port. This is the low-cost solution found by our algorithm and relies on LICCA to perform data layout, along with the alignment layers before the inputs and outputs. It is cheaper than the 8x8 Non-clustered RF, and is generally more scalable.

Table 1.

Tabular description of the connections between memory read ports and DCT logic input ports for the 16 memory accesses. The decoder logic can be derived directly from this table, using any of numerous tabular logic optimization methods.

Access	Row/Col	M1	M2	M3	M4	M5	M6	M7	M8
0	Row	1 → 1	2 → 2	3 → 3	4 → 4	5 → 5	6 → 6	7 → 7	8 → 8
1	Row	1 → 2	2 → 1	3 → 4	4 → 3	5 → 6	6 → 5	7 → 8	8 → 7
2	Row	1 → 3	2 → 4	3 → 1	4 → 2	5 → 7	6 → 8	7 → 5	8 → 6
3	Row	1 → 4	2 → 3	3 → 2	4 → 1	5 → 8	6 → 7	7 → 6	8 → 5
4	Row	1 → 5	2 → 6	3 → 7	4 → 8	5 → 1	6 → 2	7 → 3	8 → 4
5	Row	1 → 6	2 → 5	3 → 8	4 → 7	5 → 2	6 → 1	7 → 4	8 → 3
6	Row	1 → 7	2 → 8	3 → 5	4 → 6	5 → 3	6 → 4	7 → 1	8 → 2
7	Row	1 → 8	2 → 7	3 → 6	4 → 5	5 → 4	6 → 3	7 → 2	8 → 1
8	Col	1 → 1	2 → 2	3 → 3	4 → 4	5 → 5	6 → 6	7 → 7	8 → 8
9	Col	1 → 2	2 → 1	3 → 4	4 → 3	5 → 6	6 → 5	7 → 8	8 → 7
10	Col	1 → 3	2 → 4	3 → 1	4 → 2	5 → 7	6 → 8	7 → 5	8 → 6
11	Col	1 → 4	2 → 3	3 → 2	4 → 1	5 → 8	6 → 7	7 → 6	8 → 5
12	Col	1 → 5	2 → 6	3 → 7	4 → 8	5 → 1	6 → 2	7 → 3	8 → 4
13	Col	1 → 6	2 → 5	3 → 8	4 → 7	5 → 2	6 → 1	7 → 4	8 → 3
14	Col	1 → 7	2 → 8	3 → 5	4 → 6	5 → 3	6 → 4	7 → 1	8 → 2
15	Col	1 → 8	2 → 7	3 → 6	4 → 5	5 → 4	6 → 3	7 → 2	8 → 1

X → Y indicates a connection from memory read port X to DCT logic input port Y.

Memory access i has identical connectivity to memory access i+8.

Single- and Dual-Port Memories: A 64-entry single-(dual-) port flip-flop memory; most 64-entry flip-flop memories are smaller than SRAM memories, or are, at most, marginally larger.

The single and dual port memories cannot match the bandwidth of the two register files; consequently, the accesses to the memories must be serialized, and the DCT logic will require several cycles to execute. These memories are sub-optimal from the perspective of performance, and would not be considered by our CAD flow due to bandwidth limitations; they are considered here for the purpose of comparison only. The Faraday Memory Compiler cannot generate memories with more than two ports; we imposed similar constraints on the flip-flop memories used in this study.

Table 2 shows the respective area of each of the four memory organizations. As one would expect, the 8x8 Non-clustered RF has the largest overall area due to eight 64:1 multiplexors (read ports) and each of the 64 registers is preceded by an 8:1 multiplexor (write ports). The area for the 8x8 Clustered RF does not include the cost of the alignment layers, as these are specific to the ISEs that we generate. The number of storage cells was identical (2048) for all memory organizations, regardless of the slight differences (<5%) in terms of storage resource area.

The areas reported in Table 2 (and the experiments that follow) were calculated for all memory organizations clocked at 150Mhz. Synthesis was performed using Synopsys Design Compiler with the goal to minimize area.

6.3 Processor Architectures

Fig. 3 depicts two the processor architectures which we used to evaluate AVS-enhanced ISEs using the different memory system organizations listed in Table 2.

As in previous work [2, 5], DMA is used to move data structures (arrays) between main memory and the AVS; the ISE logic can also read/write data from/to the base processor's register file. The processor in Fig. 3(a) is sufficient when the AVS is implemented as an 8x8 Non-clustered RF or a single/dual ported memory; to support an AVS implemented as an 8x8 clustered RF, the ISE must be augmented with alignment layers and the corresponding decoders, as shown in Fig. 3(b).

For our experiments, we synthesize and compare only the AVS and ISE logic (including the alignment layers and decoders to evaluate the 8x8 clustered RF). We are not concerned with whole system performance; Kluter et al. [5] have already shown the benefits of including AVS-enhanced ISEs to accelerate the 2D-DCT for JPEG using an 8x8 non-clustered RF implementation of the AVS. In terms of cycle count, our system would perform exactly the same. The use of an 8x8 clustered RF would primarily serve to reduce area, and possibly increase the clock frequency. We do not compare directly with Kluter et al. because their evaluation platform was an FPGA-based soft processor system, while ours is a standard cell-based ASIC design flow.

6.4 8x8 Clustered and Non-Clustered RFs

Here we compare the 8x8 clustered and non-clustered RF implementations for 2D-DCT and 2D-FFT. Both implementations include ISE logic, and the clustered implementation includes the alignment layers and alignment decoder circuitry. The ISE is designed to execute in 1 cycle. The results are shown in Table 3.

The clustered RF achieves the target clock frequency of 150 MHz, while the non-clustered RF cannot (it achieves 140 MHz in both cases). If the critical paths of the respective designs include the AVS memory and ISE logic, then the processor that uses the clustered RF will achieve a higher clock frequency; if the critical path occurs elsewhere in the design, then the respective frequencies will be the same. Table 3 also shows that the clustered RF implementation is significantly smaller than the non-clustered implementation, even when the costs of the two alignment layers and decoder circuitry are taken into account.

Table 2.

Area of different memories considered in our study.

Memory Organization	Area (mm^2)		
	Storage Resources	Combinational Logic	Total
8x8 Non-clustered RF	0.115	0.487	0.602
8x8 Clustered RF	0.110	0.074	0.184
64 entry 1-Port Memory	0.110	0.069	0.179
64-entry 2-Port Memory	0.115	0.122	0.237

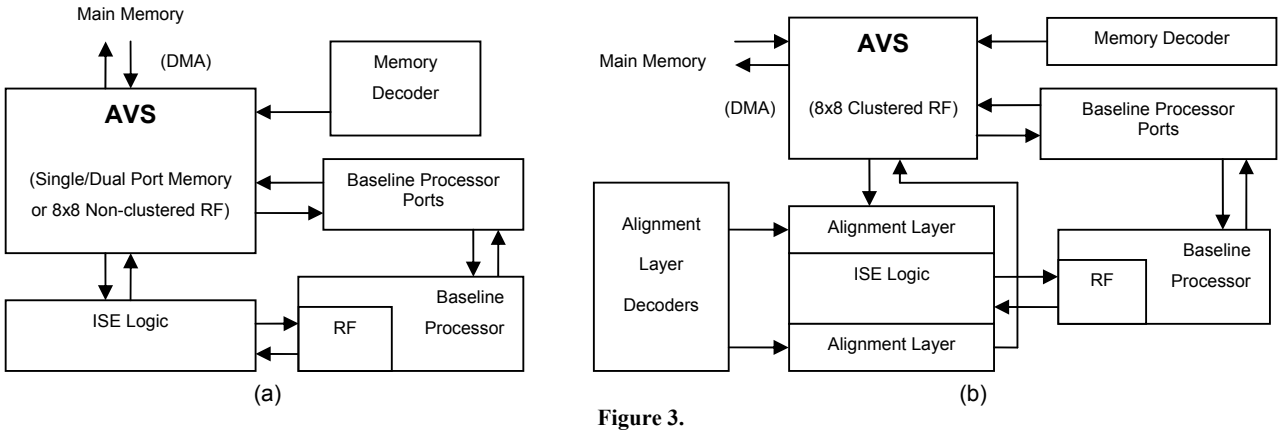

Figure 3.

Two processor organizations evaluated here; the one on the left (a) is used for 8x8 Non-clustered RF and the single and dual port memories; the one on the right (b) includes alignment layers and decoders to facilitate the 8x8 Clustered RF.

Table 3.

Comparison of the 8x8 clustered and non-clustered RF designs for the 2D-DCT and 2D-FFT benchmarks. The clustered RF achieves a higher clock frequency than the non-clustered RF and requires less silicon area. Except for the discrepancy in frequency, the performance of both memory organizations is the same, since both provide an AVS memory with 8 read and 8 write ports.

2D-DCT		
Memory Organization	Frequency	Area
8x8 Clustered RF	150 MHz	0.732 mm^2
8x8 Non-clustered RF	140 MHz	1.137 mm^2

2D-FFT		
Memory Organization	Frequency	Area
8x8 Clustered RF	150 MHz	0.684 mm^2
8x8 Non-clustered RF	140 MHz	1.097 mm^2

This experiment demonstrates that high-bandwidth, low-cost AVS memory structures can be found on an application-specific basis using a combination of data layout (via LICCA) and the appropriate use of alignment layers. The resulting memory systems are faster and cheaper than individual local memories that achieve comparable bandwidth using a large number of I/O ports.

6.5 8x8 Clustered RFs and Lower-Bandwidth Single/Dual Port Memories

Table 4 compares the 8x8 clustered RF with lower-cost alternatives that provide less I/O bandwidth to the ISE. The use of lower bandwidth memories has two effects. Firstly, the I/O accesses must be serialized: as described by Pozzi and Ienne [10], pipeline registers must be inserted into the ISE logic, which, in turn, increases the latency of the design. Secondly, reducing the I/O bandwidth reduces the size of the ISE. The 8x8 clustered RF provides 8 inputs to the ISE logic; so, a sufficient amount of logic is required to operate on 8 pieces of data in parallel. In contrast, when a single or dual port memory is used, considerably less logic is required, as one or two pieces of data are read per cycle.

For the DCT computation, a single-port memory suffices, yielding a 2-cycle ISE (that must be called several times to accomplish a single 1D-DCT operation). The use of butterfly circuits in the FFT favors a dual-ported memory. With a dual ported memory, only a single butterfly unit can be used; as the 8x8 clustered RF provides 8 read and 8 write ports, the ISE logic contains 12 butterfly units: 3 rows of 4, as shown in Fig. 4.

Table 4 compares the different memory system organizations. In terms of latency the 2D-DCT, the 8x8 clustered RF yields an ISE that is 8x faster than using smaller ISEs in conjunction with a single-port memory; for the 2D-FFT, the 8x8 clustered RF yields an ISE, shown in Fig. 4, that is 12x faster than using smaller ISEs in conjunction with a dual-ported memory. All four designs considered here achieved the target clock frequency of 150 MHz.

In terms of areas, the systems using single- and dual-port memories were considerably smaller than the one using the 8x8 clustered RF, for the reasons outlined above. Additionally, we should recall that the system using the 8x8 clustered addition to the ISE, employed input and output alignment layers and decoder circuitry for the two layers.

6.6 Runtime

For both benchmarks the cheapest memory organization (8x8 clustered RF) had a feasible layout, which LICCA discovered; the runtime of our CAD flow was approximately one minute; this is significantly faster than the runtime required to find an 8-input 8-output ISE using most existing techniques [3, 9, 12]; this runtime is comparable to the runtime of the ISE identification method of Verma et al. [13], which, to the best of our knowledge, is the fastest optimal algorithm published to date for this problem.

Table 4.

Comparison of the 8x8 clustered RF designs for the 2D-DCT and 2D-FFT benchmarks with a single- and dual-port memory. The 8x8 clustered RF-based design has a significantly larger area, in part because its larger bandwidth permits the use of larger ISEs with greater parallelism than the memories. The use of the 8x8 clustered RF yields an ISE that is 8x faster than the use of several ISEs with a single-ported memory for the 2D-DCT, and an ISE that is 12x faster than the use of several ISEs for a dual-ported memory for the FFT. All designs achieved a clock frequency of 150MHz.

2D-DCT					
Implementation	Memory Organization	I/O Ports	Area	Latency	Frequency
2D-DCT	8x8 Clustered RF	8 read, 8 write	0.732 mm^2	16 cycles	150 MHz
2-point DCT	Single-port Memory	1 read/write	0.235 mm^2	128 cycles	150 MHz

2D-FFT					
Implementation	Memory Organization	I/O Ports	Area	Latency	Frequency
2D-FFT (12 butterfly units)	8x8 Clustered RF	8 read, 8 write	0.684 mm^2	16 cycles	150 MHz
FFT (1 butterfly unit)	Dual-port Memory	2 read/write	0.270 mm^2	192 cycles	150 MHz

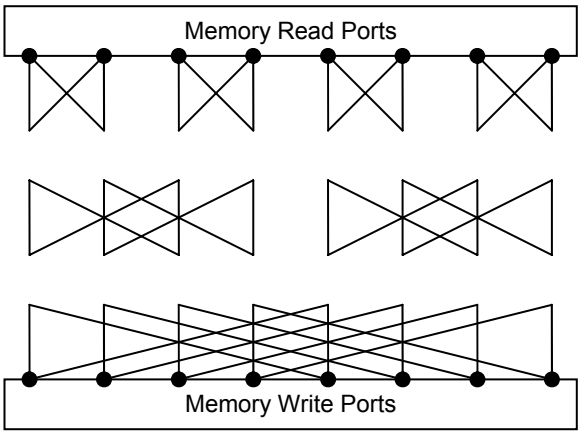

Figure 4.

ISE implementing the 2D-FFT operation; with 8 read ports and 8 write ports available, 3 rows of 4 butterfly units can perform the FFT in a single cycle.

7. RELATED WORK

Numerous algorithms to automatically identify and synthesize ISEs from a compiler's intermediate representation of a program have been proposed within the last decade [1-5, 9, 10, 12, 13]. In the context of our work, the ISE identification algorithm used is immaterial, as long as it can handle identify AVS-enhanced ISEs.

7.1 AVS-Enhanced ISEs

Biswas et al. [1] introduced the notion of state-holding ISEs, which permitted certain values to reside in registers; they followed this up with a more comprehensive approach to AVS [2]: each array that is read/written by an ISE is stored in a local memory; the compiler schedules DMA transfers to movie data between main memory and the AVS memory; and load and store operations that access the AVS memory are permitted for inclusion in ISEs. The AVS memory is not a cache, in the sense that it is placed under compiler control: in principle, the compiler knows precisely which data resides in each AVS memory at each point during program execution. The compiler can also recognize when two arrays' lifetime in the AVS is mutually exclusive, permitting them to be stored in the same memory.

Kluter et al. [5] observed that the unrestricted use of AVS creates a coherence problem between the local memories and data in the L1 cache; their solution was to use a MESI states L1 cache coherence protocol to maintain coherence.

The aforementioned prior work is the starting point for this paper: it has established the feasibility and utility of AVS-enhanced ISEs; however, the issue of memory layout in this context, and the interface between the local memories and the ISE datapath has not been considered until now. This paper, therefore, can be viewed as an enhancement to the synthesis of AVS-enhanced ISEs that can reduce the cost of the design compared to expensive multi-ported memories and without compromising bandwidth.

The 8x8 clustered RF shares some principle similarities with the work of Karuri et al. [4], who proposed the use of clustered VLIW-style register files as a means to increase data bandwidth to ISEs. Their work should not use AVS, however, as their innovations involve the processor's register file rather than storage resources that are local to the ISE logic.

7.2 Memory Optimization and Layout Techniques for Embedded Systems

Numerous techniques for memory layout and memory design space exploration in the context of compilers for embedded processors and high-level synthesis systems have been proposed in the past. For example, Schmit and Thomas [11] presented an automatic synthesis tool that can find low-cost memory designs; their goal, in contrast to ours is to find a good memory choice for a collection of arrays. Although we also search for a low-cost memory, we also take access patterns into account and use this information to find an effective layout of data in the memory.

Panda et al. [7] present a comprehensive survey of memory optimizations used in embedded systems. In particular, their discussion of "memory packing and array-to-memory assignment" [7, Section 3.3] is similar in principle to the LICCA problem formulation; one important difference is that they consider reordering parallel memory accesses through instruction scheduling; we do not have the leeway to perform this reordering as our concurrent memory accesses occur in the context of ISE invocations; in general, we cannot move an access to an array element from one ISE invocation to the next without altering the computation performed by the program.

The data layout problem, formulated here as LICCA, is a generalization of the connectivity binding problem for commutative operators in high-level synthesis, which was proven NP-complete for binary operators by Pangrle [8]. Pangrle's problem formulation is a degenerate case of LICCA, where there are only two single-ported memories without capacity restrictions. In our context, the alignment layer that we propose allows the ISE to be treated as a commutative operation, regardless of the computation it performs.

8. EXTENSIONS AND FUTURE WORK

One potential area for future work is to generate memories and data layouts for a set of ISEs with conflicting access patterns that operate on the same data. An example is shown in Fig. 5.

The input is a 4x4 matrix. The first ISE accesses the matrix using a sliding 2x2 window, which is representative of motion estimation algorithms in video decoding. The data layout, in this case, is an established data layout pattern called Morton Layout [6], as shown in Fig. 5(a). Now, suppose that the second ISE has a row access pattern. A similar organization that can facilitate efficient row accesses is shown in Fig. 5(b).

Unfortunately, a hypothetical third ISE that requires column accesses would require a costly multi-ported memory with 4 read and 4-write ports in order to access each column in a single-cycle. With a limitation of four read and four write ports, this high cost solution is simply the best we can do, given the memory access constraints of the three ISEs in question.

On the other hand, if we search for memory organizations where the number of available memory ports exceeds the bandwidth of each individual ISE, it may be possible to find low-cost solutions that do not require multi-ported memories through the use of a well-designed alignment layer. An example is shown in Fig. 5(d), which uses five distinct memories, each with one read and one write port.

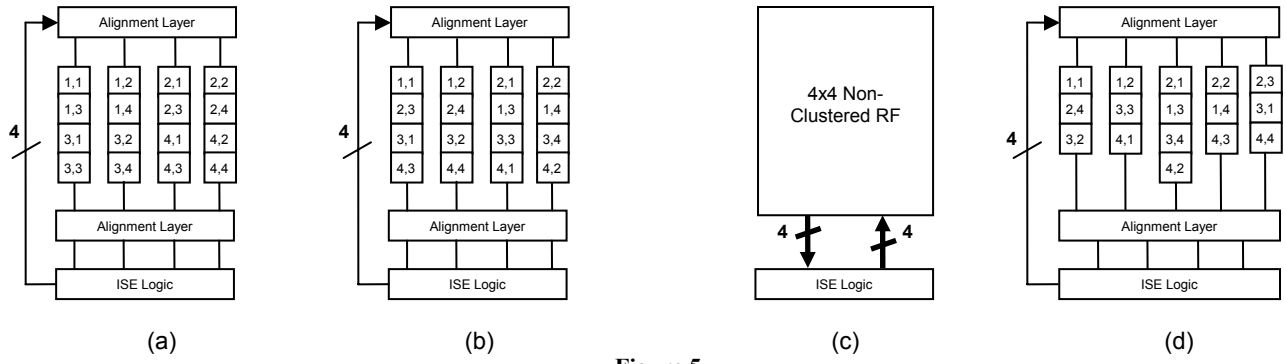

Figure 5.

Morton Layout supporting a 2x2 sliding window operation (a); modified Morton Layout supporting 2x2 sliding window operations and row accesses; a 4x4 non-clustered RF is required to support 2x2 sliding window operations along with both row and column accesses (c); however, if we increase the number of I/O ports to 5, we can support all three access patterns with 5 single-ported memories and appropriate alignment layers (d).

This is not the lowest-cost solution for a single-ISE, but is perfectly reasonable given the set of ISEs that we consider here. The possibility and overhead of ignoring such conflicts in order to avoid resorting to costly and complex memory organizations is left open for future work.

9. CONCLUSION

This paper has presented a systematic technique to reduce the silicon area required for AVS-enhanced ISEs without compromising I/O bandwidth. The technique combines a search for the lowest cost memory system organization, followed by a data layout phase (formulated as LICCA—a problem akin to graph coloring), and the use of input and output alignment layers placed between the memory system and ISE logic. Optimizing the memory subsystem using this approach reduces the silicon area by around 36% while maintaining the same data bandwidth as a multi-port memory, and without clock frequency degradation.

REFERENCES

[1] Biswas, P., Choudhary, V., Atasu, K., Pozzi, L., Ienne, P., and Dutt, N. 2004. Introduction of local memory elements in instruction set extensions. In *Proceedings of the 41st Annual Conference on Design Automation* (San Diego, CA, USA, June 07 - 11, 2004). DAC '04. ACM, New York, NY, 729-734. DOI= http://doi.acm.org/10.1145/996566.996765.

[2] Biswas, P., Dutt, N., Pozzi, L.. and Ienne, P. Introduction of architecturally visible storage in instruction set extensions. *IEEE Trans. CAD* 26, 3 (Mar. 2007), 435-446. DOI = http://dx.doi.org/10.1109/TCAD.2006.890582.

[3] Clark, N., Zhong, H., and Mahlke, S. Automated custom instruction generation for domain-specific processor acceleration. *IEEE Trans. Computers* 54, 10 (Oct. 2005), 1258-1270. DOI= http://doi.ieeecomputersociety.org/10.1109/TC.2005.156.

[4] Karuri, K., Chattopadhyay, A., Hohenauer, M., Leupers, R., Ascheid, G., and Meyr, H. 2007. Increasing data-bandwidth to instruction-set extensions through register clustering. In Proceedings of the 2007 IEEE/ACM international Conference on Computer-Aided Design (San Jose, California, November 05 - 08, 2007). International Conference on Computer Aided Design. IEEE Press, Piscataway, NJ, 166-171. DOI= http://doi.acm.org/10.1145/1326073.1326108.

[5] Kluter, T., Brisk, P., Ienne, P., and Charbon, E. 2008. Speculative DMA for architecturally visible storage in instruction set extensions. In *Proceedings of the 6th IEEE/ACM/IFIP international Conference on Hardware/Software Codesign and System Synthesis* (Atlanta, GA, USA, October 19 - 24, 2008). CODES/ISSS '08. ACM, New York, NY, 243-248. DOI= http://doi.acm.org/10.1145/1450135.1450191.

[6] Morton, G. M., 1966. A computer oriented geodetic data base; and a new technique in file sequencing. *Technical Report*, IBM Ltd., Ottawa, Canada.

[7] Panda, P. R., Catthoor, F., Dutt, N. D., Danckaert, K., Brockmeyer, E., Kulkarni, C., Vandercappelle, A., and Kjeldsberg, P. G. 2001. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.* 6, 2 (Apr. 2001), 149-206. DOI= http://doi.acm.org/10.1145/375977.375978.

[8] Pangrle, B. M. On the complexity of connectivity binding. *IEEE Trans. CAD* 10, 11 (Nov. 1991), 1460-1465. DOI= http://doi.ieeecomputersociety.org/10.1109/43.97625.

[9] Pozzi, L., Atasu, K., and Ienne, P. Exact and approximate algorithms for the extension of embedded processor instruction sets. *IEEE Trans. CAD* 25, 7 (Jul. 2006), 1209-1229. DOI= http://doi.ieeecomputersociety.org/10.1109/TCAD.2005.855950.

[10] Pozzi, L. and Ienne, P. 2005. Exploiting pipelining to relax register-file port constraints of instruction-set extensions. In Proceedings of the 2005 international Conference on Compilers, Architectures and Synthesis For Embedded Systems (San Francisco, California, USA, September 24 - 27, 2005). CASES '05. ACM, New York, NY, 2-10. DOI= http://doi.acm.org/10.1145/1086297.1086300.

[11] Schmit, H., and Thomas, D. E. Synthesis of application-specific memory designs. *IEEE Trans. VLSI* 5, 1 (May 1997) 101-111. DOI = http://doi.ieeecomputersociety.org/10.1109/92.555990.

[12] Sun, F., Ravi, S., Raghunathan, A., and Jha, N. K. Custom-instruction synthesis for extensible processor platforms. *IEEE Trans. CAD* 23, 2 (Feb. 2004), 216-228. DOI= http://doi.ieeecomputersociety.org/10.1109/TCAD.2003.822133.

[13] Verma, A. K., Brisk, P., and Ienne, P. 2007. Rethinking custom ISE identification: a new processor-agnostic method. In *Proceedings of the 2007 international Conference on Compilers, Architecture, and Synthesis For Embedded Systems* (Salzburg, Austria, September 30 - October 03, 2007). CASES '07. ACM, New York, NY, 125-134. DOI= http://doi.acm.org/10.1145/1289881.1289905.