

High-Level Synthesis with SIMD Units

Vijay Raghunathan, Anand Raghunathan[†], Mani B. Srivastava, and Milos D. Ercegovac[‡]

Department of Electrical Engineering, UC Los Angeles, CA 90095

[†]NEC-USA, C&C Research Labs, Princeton, NJ 08540

[‡]Department of Computer Science, UC Los Angeles, CA 90095

Abstract—This paper presents novel techniques to integrate the use of Single Instruction Multiple Data (SIMD) functional units in a high-level synthesis (HLS) design methodology. SIMD functional units can be configured to operate in one or more SIMD modes, in which they process multiple sets of smaller bitwidth operands in parallel. Conceptually, the use of SIMD functional units enables HLS to (i) exploit parallelism to a higher degree without using additional resources, (ii) improve resource utilization by enabling hardware re-use at a fine-grained level, and (iii) improve energy efficiency for a given area and/or performance constraint.

We illustrate the issues involved in performing high-level synthesis with SIMD functional units, and discuss how algorithms involved in a typical high-level synthesis flow can be enhanced to result in maximal performance and energy improvements. These techniques are not restricted to specific high-level synthesis tools/algorithms, and can be plugged into any generic high-level synthesis system. Experimental results indicate that, the use of SIMD units can improve performance by up to 1.9X (average of 1.57X), and simultaneously reduce energy consumption by up to 33.16% (average of 28.03%) compared to well-optimized conventional designs, with minimal area overheads (average of 2.18%). The performance improvements can be translated into additional energy savings, resulting in upto 66.26% (average of 55.88%) energy reductions. Further, our experiments demonstrate that, the use of SIMD units in a HLS tool results in a shift in the entire area-delay-energy tradeoff envelope that can be obtained, to include desirable parts of the design space (*i.e.*, higher quality designs) that were hitherto unreachable.

I. INTRODUCTION

SEVERAL application domains (*e.g.*, multimedia processing, signal processing, network protocol processing), are characterized by a combination of computational features like fixed-point arithmetic, a variety of short data types, and a high degree of fine-grained operation-level parallelism. The implementation of these algorithms frequently comprises of variables and computations whose precision (*i.e.*, bitwidth) varies significantly. These characteristics enable the use of Single Instruction Multiple Data (SIMD) functional units during the hardware/software implementation of these algorithms. Unlike conventional functional units, SIMD functional units can be dynamically configured to operate in one or more SIMD modes, in which they perform the same operation on several sets of smaller bitwidth data in parallel. The general principles of SIMD computation have been well researched in areas such as high performance computer architecture, parallel computing, advanced compiler design, and digital arithmetic, which are reviewed in Section I.B. The potential benefits offered by SIMD computation has been described in the above areas of research, mostly in the context of (manually designed) custom integrated circuits [1], [2], [3]. The success of these techniques, together with the need to bridge the quality gap that exists between custom designed and synthesized circuits [4], motivates us to explore their use in the context of high-level ASIC design tools.

A. Paper Overview and Contributions

In this paper, we exploit techniques employed in digital arithmetic and custom processor design, to incorporate the use of SIMD functional units into automatic high-level synthesis (HLS). We demonstrate

that the proposed techniques are capable of enhancing the design space explored by a high-level synthesis tool, resulting in designs that are superior with respect to various design metrics. In general, application of our techniques results in an expansion of the area-delay-energy trade-off envelope that characterizes any given high-level synthesis tool, to include higher quality designs.

We present simple, yet powerful, hardware architectures for performing SIMD computations on variable precision data, and illustrate the performance improvements and energy savings that can be obtained through the use of these SIMD functional units during high-level design. Further, we demonstrate that naive use of SIMD units does not fully exploit their potential, and can sometimes lead to sub-optimal designs. We discuss how algorithms for the various steps involved in a typical high-level synthesis tool can be further enhanced to result in maximal performance and energy improvements.

Experimental results in the context of a commercial design flow demonstrate that the proposed SIMD unit based synthesis techniques achieve significant performance improvements and simultaneous energy savings over conventional high-level synthesis techniques. These results testify to the high potential of the SIMD computation paradigm in the synthesis of high-performance and/or low-power application specific hardware architectures.

B. Related Work

Multiple precision computation has been an active research topic in computer arithmetic for a long time. Variable precision algorithms for multiplication, and square-root have been proposed in [5], [6]. Recent general purpose architectures like the Intel MMX extension to the Pentium [1], the SUN UltraSparc II Visual Instruction Set architecture [2], and the HP MAX2 [3] provide instruction sets and architectural support for the parallel execution of low-precision instructions on partitioned arithmetic logic units. Techniques for efficient software utilization of these micro-architectures have been proposed in [7], [8], [9].

Conventional HLS tools use techniques such as chaining, multi-cycling, and pipelining of components to obtain performance and power improvements [10], [11]. However, the atomic components used for constructing the register transfer level (RTL) circuits are assumed to operate on a single set of operands at a time, *i.e.*, single instruction single data (SISD) type of computation. High-level synthesis techniques for area and power optimization, using multiple precision arithmetic units are presented in [12], [13]. A multiple precision functional unit is one that can be re-used over multiple cycles to perform a higher bitwidth operation than the unit's maximum bitwidth. In contrast, SIMD units utilize unused portions of hardware modules to execute multiple instances of the same operation type in parallel, on smaller bitwidth data. Precision sensitive scheduling and resource sharing techniques were proposed in [14], and shown to be effective in reducing area through the use of conventional, non-SIMD units. In summary, the benefits obtained through the use of SIMD units are complementary to the techniques proposed in the related work above.

The rest of this paper is organized as follows. Section II introduces the operation of SIMD functional units, illustrates their potential benefits, and demonstrates that realizing these benefits requires judicious integration into high-level synthesis. Section III presents examples of SIMD unit design, and highlights the architectural issues involved in designing SIMD functional units. Section IV describes how algorithms for the various steps involved in a typical high-level synthesis tool can

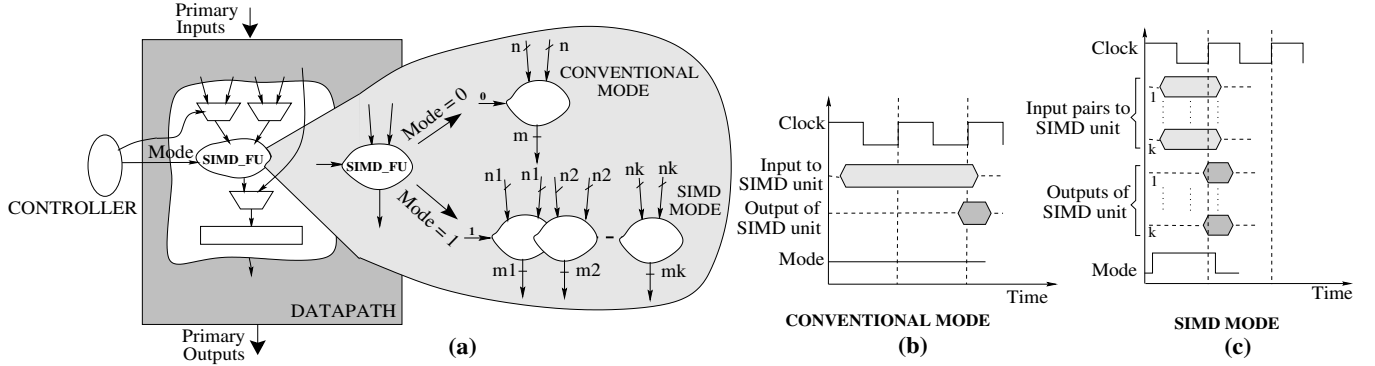


Figure 1. (a) An SIMD functional unit embedded in an abstract RTL circuit (b) Symbolic waveforms illustrating the timing behavior of an SIMD functional unit in the conventional mode of operation (c) Timing behavior of the same functional unit in SIMD mode of operation

be enhanced to result in maximal performance and energy improvements. Section V presents experimental results, that demonstrate the improvements resulting from the proposed techniques.

II. MOTIVATION

SIMD components exhibit the property that they can be dynamically configured to execute multiple smaller bitwidth operations in parallel. The operation of an example SIMD component embedded in an abstract RTL controller/datapath circuit is illustrated in Figure 1(a). In addition to the input operands for the computation, the SIMD functional unit has another input “Mode” (provided by the controller), that specifies its operating mode. A logic value of 0 on the Mode signal results in the functional unit operating as a conventional n -bit module. When the Mode signal assumes a logic value 1, the same functional unit performs k operations in parallel on input operands that are $n_1, n_2, \dots, n_k$ bits wide (where $\sum_{i=1}^k n_i = n$), as illustrated in Figure 1(a). The symbolic waveforms in Figures 1(b) and 1(c) illustrate the operation of an SIMD functional unit in the conventional and SIMD modes of operation, respectively.

The use of SIMD units results in higher quality designs due to the following factors:

- SIMD components offer a way to alleviate the limits on parallelism imposed by resource constraints, by performing multiple operations in parallel on a single resource. In effect, the use of SIMD functional units improves resource utilization since previously unused parts of a component are now used to perform additional computations. For example, if a conventional 16-bit adder is used to perform an 8-bit addition, the higher order bit-slices of the adder are not utilized for the operation. On the other hand, an SIMD adder can use the higher order bit-slices to perform a second 8-bit addition in parallel, thus fully utilizing the computational capability of the hardware. In general, the higher degree of parallelism enabled by SIMD units leads to lower execution time for a given area or lower area for a given execution time.
- SIMD components have a variable latency of computation. The time (number of clock cycles) taken to compute the outputs depends on the mode of operation of the component. For example, the time taken by a 32-bit SIMD adder implemented in the HP14B 0.5 μ m technology [15] to perform two 16-bit additions in parallel is 6.51ns, which is much lower than the time taken to perform one 32-bit addition, which is 12.68ns. Thus, under a clock period of 14ns, the SIMD adder takes two clock cycles to compute the output while operating in conventional mode, and only one clock cycle to compute the output while operating in SIMD mode. This variable latency of SIMD units leads to a further increase in performance.

In addition to performance improvements, the use of SIMD functional units also leads to a significant decrease in the energy consumption of a design, due to the following factors:

- The total capacitance switched by a functional unit per operation is approximately the same even if executes multiple smaller

bitwidth operations in parallel. For example, a conventional 32-bit adder implemented in the HP14B 0.5 μ m technology [15], switches around 17.77pF of capacitance to perform one addition, while an SIMD adder can perform two 16-bit additions while switching approximately the same amount of capacitance. Hence, by packing multiple smaller bitwidth operations into a single operation, SIMD functional units reduce the total amount of capacitance switched during the execution of the computation, which translates into a reduction in energy consumption.

- A reduction in the number of clock cycles to complete the design’s computation translates into a reduction in the total switched capacitance in the clock network and registers.
- The performance improvement obtained with respect to a fixed-precision architecture could be traded off for further reduction in energy consumption through supply voltage scaling or other performance-power tradeoff techniques.

The following example illustrates the performance improvement and energy savings that can be obtained through the use of SIMD functional units during high-level synthesis.

Example 1: Consider the behavior fragment shown in Figure 2(a). The behavior represents the body of a loop that computes the expressions $(x5-(x1*x2))*17$ and $(x3*x4)+x2+x3+255$, and writes the results into two arrays in memory. The library of components that can be used to implement the behavior is shown in Figure 2(b). The library includes conventional, fixed-precision components as well as SIMD components. Suppose the designer wants to implement the behavior of Figure 2(a) under the resource constraint: 1 LT_CMP1, 1 ADD1, 1 SUB1, 1 MUL1. Further, suppose that all the primary inputs are 8 bits wide and the result of all arithmetic operations that are larger than 16 bits are truncated to 16 bits. The bitwidths of all the variables are annotated in Figure 2(a). Note that operation *3 has one 16-bit input and hence, the multiplier has to have a minimum bitwidth of 16. We generated RTL implementations of this CDFG without and with the use of SIMD functional units. Figure 2(c) shows the schedule for an implementation that uses only fixed precision functional units, in state transition graph (STG) format. From the CDFG, it is clear that operations *1 and *2 do not have any dependencies and thus, can be performed in parallel. However, since there is only one multiplier available, they are forced to execute sequentially as shown in the STG of Figure 2(c). Instead, if the multiplier is of type SIMD_MUL1, it can operate in SIMD mode on two sets of 8-bit operands simultaneously. Thus, operations *1 and *2 can indeed be performed in parallel, as shown in the STG of Figure 2(d). A similar argument applies to operations +1 and +4, which are mapped to component SIMD_ADD1.

Upon examining the two STGs, it can be seen that, in the schedule of Figure 2(c), each iteration of the loop requires 7 cycles, while in the schedule of Figure 2(d), each loop iteration requires 5 cycles. In order to compare the performance, area, and energy consumption of both designs, the generated RTL implementations were synthesized using Synopsys Design Compiler [16] and mapped to the HP14B 0.5 μ m technology library [15]. We calculated the *expected execution time* (EET) metric to compare the performance of the two designs. The

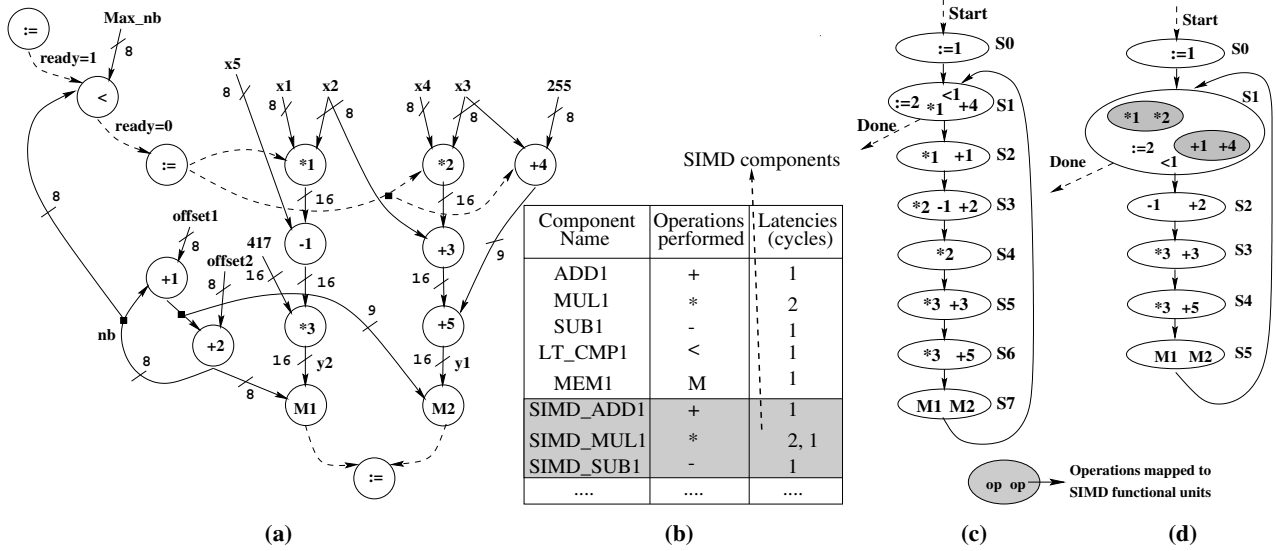


Figure 2. (a) An example behavior, (b) RTL component library (c) A schedule for the behavior (d) An alternate schedule for the same behavior, using SIMD components

EET of a design is the product of the *expected number of clock cycles* (ENC)¹ [17] and the clock period. The schedule of Figure 2(c) has an EET of 2218.22ns, while the schedule of Figure 2(d) has an EET of 1621.96ns, which represents a performance improvement of 1.37X. This is because: (i) the use of the SIMD multiplier speeds up the path ($*2 \rightarrow +3 \rightarrow +5 \rightarrow M2$), since operation $*2$ now starts in state S1 instead of state S3, speeding up the loop body in the process. Similarly, operation $+1$ is advanced from state S2 to S1 due to the use of the SIMD adder, and (ii) the multiplier SIMD_MUL1, operates in SIMD mode in state S1 (i.e., operations $*1$ and $*2$) and hence has a latency of one clock cycle compared to a two cycle latency in conventional mode, resulting in further performance improvement.

The use of SIMD functional units results in a significant reduction in energy consumption. The implementation corresponding to the schedule of Figure 2(c) was found to consume 11.19nJ per iteration, while the schedule of Figure 2(d) was found to consume 7.99nJ per iteration, which represents an energy savings of 50.77%. Note that, the above savings represent *simultaneous performance and energy improvements*. The performance improvements can be translated into further savings in energy consumption through supply voltage scaling or other performance-power tradeoff techniques. For example, scaling the supply voltage of the optimized design till it has the same performance as the original design results in an energy savings of 50.77%. These improvements come at the cost of a very small area overhead of 2.78%, which is due to the configuration and control circuitry needed to enable SIMD operation of the functional units.

The above example illustrates that the use of SIMD functional units enables significant simultaneous performance and energy improvements. However, SIMD functional units need to be judiciously integrated into a high-level synthesis flow to fully realize their potential. The next example illustrates that the manner in which these functional units are integrated into high-level synthesis can adversely impact the amount of performance improvement obtained.

Example 2: Consider the example behavior shown in Figure 3. The RTL library considered is the same as the one shown in Figure 2(b). As indicated in Figure 3, the resource constraints are 1 LT_CMP1, 1 MUL1, 1 ADD1, and 1 SIMD_ADD1. We subjected this behavior to high-level synthesis while changing the pairs of operations that are selected for SIMD execution. Across the range of resulting designs, the performance (EET) was found to vary from a worst case of 6745ns (when operations $+4$ and $+8$ were mapped to execute in the SIMD

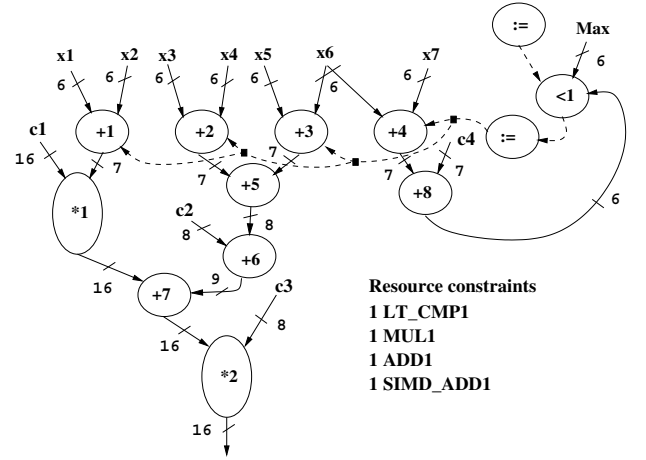


Figure 3. Behavior used in Example 2

mode) to a best case of 5783ns (when any two operations from among $+1$, $+2$, and $+3$ are mapped to execute in the SIMD mode). The best possible design without the use of SIMD functional units has a performance of 6745ns. Thus, we see that the manner in which SIMD functional units are incorporated into high-level synthesis significantly impacts the performance of the resulting implementation. Further, the selection of operations to be mapped onto SIMD functional units should consider the following guidelines.

- Operations that lie on critical paths of the CDFG should be given higher preference, since speedup of the critical paths is necessary to achieve performance improvement. For example, in the CDFG of Figure 3, mapping operations $+2$ and $+4$ onto the SIMD adder does not result in any performance improvement, since $+4$ does not lie along a critical path. Instead, if operations $+2$ and $+3$ were mapped onto the SIMD adder, and $+1$ onto the conventional adder, the critical paths ($+2 \rightarrow +5 \rightarrow +6 \rightarrow +7 \rightarrow *2$), ($+3 \rightarrow +5 \rightarrow +6 \rightarrow +7 \rightarrow *2$), and ($+1 \rightarrow *1 \rightarrow +7 \rightarrow *2$) can all execute in the minimum possible time, resulting in performance improvement - in fact, this mapping results in the lowest value of EET (5783ns) under the specified resource constraints.
- The operations that are mapped to the SIMD functional unit should be parallelizable. For example, in the CDFG of Figure 3, mapping operations $+2$ and $+5$ to the SIMD adder does not result in any performance improvement, since the two operations cannot

¹ ENC is a performance metric used for schedules of behavioral descriptions that contain significant control constructs in the form of conditionals and data-dependent loops. It can be calculated using the procedure described in [17]

TABLE I: DELAY, AREA, AND ENERGY PER OPERATION VALUES FOR CONVENTIONAL AND SIMD RTL COMPONENTS

Component	Conventional			SIMD			
	Delay (ns)	Area (μm^2)	Energy/op. (pJ)	Mode	Delay (ns)	Area (μm^2)	Energy/op. (pJ)
add_32	11.93	19620	193.5	32	12.68	20294	209.2
				16-16, 16-8-8	6.51		202.8
				8-8-8-8	3.30		199.6
sub_32	12.38	22099	213.0	32	13.02	22618	227.4
				16-16, 16-8-8	6.69		219.7
				8-8-8-8	3.45		216.6
mult_16_16	18.96	199460	2877.5	16	19.43	205815	3054.3
				8-8	9.24		3016.0
gt_cmp_32	3.67	17266	43.7	32	4.21	19368	56.3
				16-16, 16-8-8	3.32		48.8
				8-8-8-8	2.18		43.2

be executed in parallel due to the data dependency between them.

The above example showed that naive use of SIMD functional units during high-level synthesis can lead to designs that are sub-optimal in performance. For an efficient high-level design methodology that exploits the performance and energy optimization potential of SIMD functional units, the following requirements have to be fulfilled. (i) An RTL library of these SIMD functional units has to be constructed, which the high-level synthesis tool can use to generate optimized architectures. (ii) The high-level synthesis algorithms themselves have to be modified in order to generate SIMD architectures with maximum performance and energy gains, and minimum area overheads. In the following two sections, we present hardware architectures for SIMD functional units, and then describe how these units can be seamlessly integrated into a high-level design environment.

III. SIMD FUNCTIONAL UNIT ARCHITECTURE

In this section, we describe the hardware architecture of SIMD functional units. We have developed an extensive RTL library of several SIMD functional modules of various bitwidths. In order to compare these SIMD units with conventional ones with respect to delay, area and energy consumption, we synthesized the RTL implementations of these SIMD functional units as well as the corresponding conventional implementations, using Synopsys Design Compiler [16] and mapped them to the HP14B 0.5 μm technology library [15]. Table I lists the delay, area and energy consumption values for a few representative conventional and SIMD functional units.

For the sake of illustration, we describe in detail the hardware architecture of an SIMD adder. The adder architecture, shown in Figure 4(a), is a 32-bit SIMD adder comprised of four smaller blocks of 8-bit adders. Such a modular design is commonly used in arithmetic functional unit design to reduce the complexity of hardware structures. The adder has a 2-bit control signal C[1:0] that determines the operating mode. The operating modes and the corresponding control signal values are shown in Figure 4(b). Depending on the operating mode, the steering logic modules L1, L2 and L3 compute and feed the correct input carry to each submodule. For example, suppose the two-bit control signal has a logic value of 11. The output of each of the steering logic modules L1, L2, and L3, assumes a logic value 0, which is fed as a carry-in to the next adder sub-module. Further, the carry out of each 8-bit adder submodule now becomes a primary output of the 32-bit SIMD adder. Thus, the SIMD functional unit becomes completely equivalent to four 8-bit adders. We refer to this mode of operation as the *8-8-8-8 SIMD mode*. The adder architecture of Figure 4(a) (using a carry lookahead structure for the ADD-8 submodules) was synthesized, and mapped to the HP14B 0.5 μm library [15]. The delay, area, and energy consumption values for the adder architecture are given in Table I. To illustrate the variable latency operation of this SIMD adder, we also computed its worst case delay while operating in various SIMD modes. The delay of the SIMD adder, which was 12.68ns in conventional mode, was found to be 6.51ns while operating in the 16-16 or

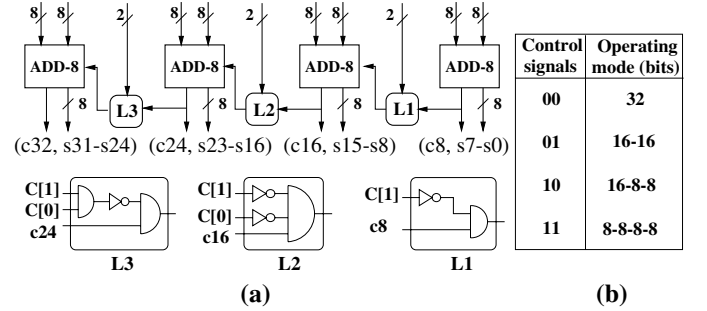


Figure 4. (a) Hardware architecture of a 32-bit SIMD adder (b) Control signals and corresponding modes of operation

16-8-8 SIMD mode². While operating in the 8-8-8-8 SIMD mode, the delay further reduced to 3.30ns. Thus, under a clock period of 14ns, if the SIMD adder takes 4 clock cycles to complete execution while operating in conventional mode, it will take only 2 clock cycles to finish execution while operating in the 16-16 and 16-8-8 SIMD modes, and 1 clock cycle to finish execution while operating in the 8-8-8-8 SIMD mode. It is worth noting that, although the extra control circuitry is on the critical path of the adder, in most cases, the resulting timing overhead after logic synthesis and technology mapping optimization is very small. As borne out by the experimental results presented in Section 5, these minimal hardware overheads in individual units enable the synthesis of significantly higher quality RTL circuits.

IV. ALGORITHM

In this section, we describe the enhancements that have to be made to a high-level synthesis flow, in order to realize the performance and energy optimization potential of SIMD functional units. Section IV.A presents an overview of the high-level synthesis framework and how the individual sub-tasks involved in high-level synthesis need to be enhanced to incorporate the use of SIMD units, while Section IV.B details the changes that have to be made to each of the constituent steps.

A. Overview

A generic high-level synthesis flow enhanced with the use of SIMD units is shown in Figure 5. The inputs are, a given behavior, resource and clock period constraints, and optimization objectives, and the output is an RTL controller/datapath architecture that uses the provided RTL component library. The various subtasks involved in high-level synthesis are: (i) compilation of the behavior into an internal representation, typically, a CDFG (ii) optimizing the CDFG by applying a series of behavioral transformations (iii) scheduling the various operations in the CDFG into control steps, thereby determining the cycle-by-cycle behavior of the resulting implementation (iv) resource sharing and allocation, which assigns the operations in the CDFG to corresponding functional units, and (v) RTL circuit generation, in which the informa-

²The critical path in the 16-16 and 16-8-8 SIMD modes is the same, resulting in equal execution times.

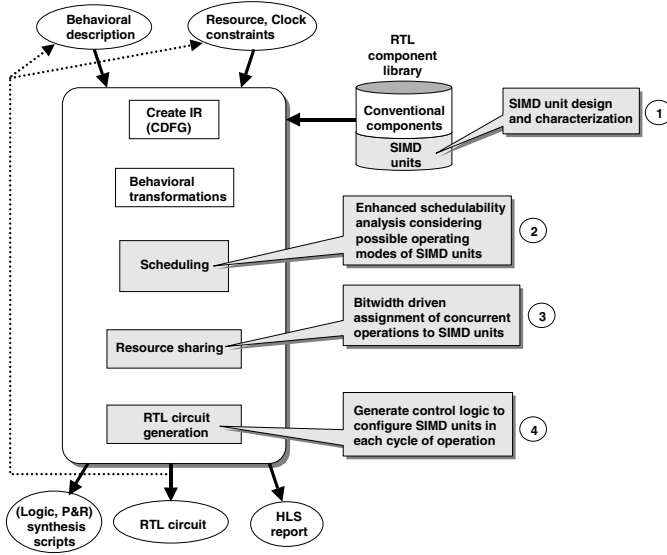


Figure 5. High-level synthesis flow with SIMD units: An overview

tion obtained during scheduling and resource sharing is used to generate the controller and datapath architectures.

The shaded boxes in Figure 5 indicate the high-level synthesis sub-tasks that need to be enhanced to incorporate the use of SIMD units into the design flow. The first step is the creation and characterization of a comprehensive RTL library of SIMD functional units, which was discussed in detail in Section III. Next, in order to exploit the increased parallelism that results from the use of SIMD units, the schedulability analysis performed during the scheduling sub-task should be modified to account for their various possible operating modes. This is detailed in Section IV.B.1. SIMD units also impact resource sharing, since multiple operations that are scheduled to execute in parallel can be assigned to the same functional unit, unlike conventional resource sharing. We describe the changes that need to be made to the resource sharing algorithm in Section IV.B.2. Finally, RTL circuit generation also changes since the controller now has to output additional signals to configure the SIMD units appropriately during each clock cycle, as described further in Section IV.B.3.

B. Details

This section details the enhancements that have to be made to a HLS framework to seamlessly integrate the use of SIMD units. Section IV.B.1 describes the scheduling phase, Section IV.B.2 details the resource sharing phase, and Section IV.B.3 talks about RTL circuit generation.

B.1 Scheduling

Figure 6 lists the pseudocode for a generic scheduler enhanced for the use of SIMD units. The scheduler accepts as inputs, the CDFG to be scheduled, resource and clock period constraints, and the module selection information. The resource constraints specify the numbers and types of functional units available for use, which may include SIMD units in addition to conventional ones. The output of the scheduler is a State Transition Graph (STG) that describes the schedule.

Pre-processing the CDFG to upscale operand bitwidths

The first step in the scheduling process involves pre-processing the CDFG to scale the bitwidth of each operand to the next higher available resource bitwidth. For example, if there are two adders available, one conventional 32-bit adder and one 16-bit SIMD adder that can also operate in the 8-8 SIMD mode, a 6-bit operand is upsampled to 8 bits, a 14-bit operand is upsampled to 16 bits, and a 26-bit operand is upsampled to 32 bits. The procedure *Pre-process_CDFG* shown in step 1 in the pseudocode of Figure 6 performs this step. After this step, all operand bitwidths exactly match the available functional unit bitwidths.

```

Scheduler (CDFG G, RESOURCE_CONSTRAINT C, CLOCK_PERIOD clk,
           MODULE_SELECTION_INFO M_inf) {
    SET<OPERATION> Unscheduled_operations,
    SET<OPERATION> Schedulable_operations,
    1 Pre-process_CDFG(G, C, M_inf);
    2 while (|Unscheduled_operations| > 0) {
        //While unscheduled operations remain
    3 op=Select_schedulable_operation(Schedulable_operations, C, M_inf, clk);
        //Select an operation for scheduling. The selected operation
        //must honor resource and clock cycle constraints. The
        //schedulability analysis is performed in this procedure.
    4 Schedule(op);
    5 Unscheduled_operations_remove_operation(op);
    6 Schedulable_operations_remove_operation(op);
    7 SET<OPERATION> schedulable_successors = Compute_sched_succ(op);
        //Find the set of operations in the fanout of op
        //which become schedulable when op is scheduled.
    8 Schedulable_operations_append(schedulable_successors);
        //Augment Schedulable_operations by addition
        //of operations in schedulable_successors.
    }
}

```

Figure 6. Pseudocode for a generic scheduling algorithm enhanced for the use of SIMD functional units

Selecting an operation to schedule in the current step

During the scheduling process, a candidate operation (*op*) from the set *Schedulable_operations* (which is the set of operations whose data and control dependencies have been satisfied and hence can be scheduled) is selected for scheduling in a given state. Candidate operations are chosen taking into account factors that affect the optimization objectives, such as existence of critical paths, *etc.* A schedulability analysis is then performed to determine whether the candidate operation is schedulable in the current state or not. For the purpose of schedulability analysis, the scheduling algorithm maintains a list of the total number of resources of each type that are still available in the current control step. The schedulability analysis verifies whether the candidate operation honors clock cycle constraints, and if at least one resource (or part of a resource, since resources can also be SIMD units) capable of executing the operation is available in the current control step. If so, it deems the operation to be schedulable. The selected operation (*op*) is then scheduled in the current state and is removed from both *Unscheduled_operations* and *Schedulable_operations*, as it no longer belongs to either of the sets. The procedure *Compute_sched_succ* identifies the operations in the CDFG that become schedulable when *op* is scheduled, which are then included in *Schedulable_operations*. The list of available resources is updated, and the scheduling continues till there are no operations remaining in *Unscheduled_operations*.

B.2 Resource sharing and allocation

The use of SIMD units enables multiple operations to share a single resource even if they have been scheduled to execute in parallel. Resource sharing has to be done carefully, with the aim of maximizing the resource utilization. The rationale behind this objective is that increasing the resource utilization leads to an increase in the amount of parallelism in the design, which in turn improves the performance. So, an operation is always mapped to the minimum bitwidth component (or part of a component in the case of an SIMD unit) that is capable of executing it and is still unused in the given control step. If there are multiple available resources of the same bitwidth, the following criterion is used to select the resource to which the operation is to be allocated. Between resources of the same bitwidth, conventional resources are always used up first. SIMD units are thus preserved, which permits the resource allocation algorithm to retain a high degree of flexibility till the very end. Note that, the mapping of operations to resources also determines the operating mode of the SIMD units. For example, if two 16-bit operations have been mapped to a 32-bit SIMD adder, then the adder has to operate in the 16-16 mode in that control step. This information can then be used to generate the appropriate mode signals for the various SIMD units on a cycle-by-cycle basis, as we describe next.

TABLE II: PERFORMANCE, AREA, AND ENERGY RESULTS

Circuit	Exec. time per iteration (ns)			Area (x1000 sq. μm)			Energy per iteration (nJ)			Energy - V_{dd} scaled (nJ)	
	Orig	SIMD	Speedup	Orig	SIMD	A.O.	Orig	SIMD	E.S.	SIMD	E.S.
MemInit	67.9	49.5	1.37X	235.4	241.9	2.78%	11.19	7.99	28.59%	5.53	50.58%
Wavelet	93.8	70.2	1.34X	37.4	37.9	1.18%	0.81	0.67	17.02%	0.47	41.97%
FIR	235.6	124.0	1.90X	253.2	259.6	2.50%	30.50	20.82	31.75%	10.29	66.26%
Add.bench	99.2	65.7	1.51X	51.4	52.3	1.70%	2.45	1.64	33.16%	1.02	58.37%
Paulin	175.3	94.3	1.86X	243.3	249.7	2.61%	22.76	16.44	27.72%	8.31	63.49%
Quad	86.6	59.8	1.45X	283.9	291.4	2.64%	24.07	16.85	29.98%	11.01	54.26%
SpHarm	102.5	60.6	1.69X	239.1	244.0	2.05%	8.78	6.50	25.97%	3.61	58.88%
PPsum	28.3	19.9	1.42X	242.8	247.6	1.98%	7.02	4.91	30.06%	3.28	53.27%

B.3 RTL circuit generation

RTL circuit generation refers to the process of constructing a data-path and controller from the information generated in the scheduling and resource sharing steps. The tasks involved in RTL circuit generation may include interconnect style and topology optimization, data path generation, and controller generation. In HLS tools that employ SIMD units, the data path generation step will instantiate conventional as well as SIMD units from the RTL component library. As described in Section II, SIMD units have additional control inputs that are used to select their operation mode. The controller generation step is enhanced to generate control logic that drives the mode select control inputs of each SIMD unit employed in the data path. The results of scheduling and resource sharing encapsulate the information necessary to determine the values of the mode select inputs in each control state, as illustrated in the following example.

Example 3: Consider again the example behavior shown in Figure 2(a), and its corresponding schedule shown in Figure 2(d) that employs two SIMD components, SIMD_ADD1 and SIMD_MUL1. As mentioned in Example 1, SIMD_ADD1 has two operating modes: a conventional 16-bit mode, and a SIMD mode where it performs two 8-bit additions. These modes are selected by setting the mode select control input, SIMD_ADD1_mode, to 0 and 1, respectively. The instance of SIMD_ADD1 is used in SIMD mode in state $S1$ of the schedule, and in conventional mode in states $S2$, $S3$ and $S4$. Hence, the specification for the control logic that generates signal SIMD_ADD1_mode is $\{S0 \rightarrow X, S1 \rightarrow 1, S2 \rightarrow 0, S3 \rightarrow 0, S4 \rightarrow 0, S5 \rightarrow X\}$, where X represents a don't care. A similar argument holds for SIMD_MUL1 and its control input SIMD_MUL1_mode. ■

V. EXPERIMENTAL RESULTS

This section describes the results of application of the proposed SIMD unit based synthesis techniques to several example benchmarks. We enhanced an existing conventional RTL library by adding SIMD implementations of various functional units, including adders, subtractors, multipliers, and different types of comparators. Using this enhanced component library, we generated structural RTL implementations for each benchmark, without and with the use of SIMD functional units, using identical resource constraints and optimization objectives for both the cases. We refer to these as the original and SIMD implementations, respectively. Logic optimizations were performed using Synopsys Design Compiler [16], and the design was technology mapped to the HP14B 0.5 μm technology library [15]. The resulting circuits were compared with respect to area, performance, and energy consumption. We evaluated the proposed techniques using the following example benchmarks. Example Meminit was discussed in Section II. Wavelet, which implements the discrete wavelet transform, and FIR, which represents an 8-tap finite impulse response filter, are well known image and signal processing algorithms. Example Add.bench implements a large set of synthetic additions. Paulin is a well known high-level synthesis benchmark. Quad is a subroutine used in the computation of Gauss-Jacobi abscissas and weights. SpHarm computes spherical harmonics for solving wave equations. PPsum is a parallel prefix sum routine used in address calculations.

Table II summarizes the results of our experiments. Major columns *Circuit*, *Exec. time*, *Area*, *Energy*, and *Energy- V_{dd} scaled* represent

the name of the design, execution time per iteration, area, energy consumed per iteration without V_{dd} scaling, and the energy consumption after V_{dd} scaling, respectively. Minor columns *Orig* and *SIMD* represent the conventional and SIMD implementations. Column *Speedup* represents the improvement in performance, column *E.S.* represents the energy savings, and column *A.O.* represents the area overhead incurred by our technique. The results indicate that the proposed SIMD functional unit based synthesis techniques achieve a performance improvement of upto 1.9X (average of 1.57X) over a state-of-the-art high-level synthesis tool. Also, the energy consumed by the synthesized design decreases by upto 33.16% (average of 28.03%). Note that the use of SIMD functional units leads to a simultaneous performance improvement and energy savings. The performance improvement can be translated into additional energy savings through performance-power tradeoff techniques like supply voltage scaling. We also calculated the energy savings that result when V_{dd} scaling is performed. Supply voltage scaling was performed using the equation $\frac{V_{dd}^{initial}}{(V_{dd}^{initial} - V_t)^2} \times T_{orig} = \frac{V_{dd}^{new}}{(V_{dd}^{new} - V_t)^2} \times T_{opt}$, where $V_{dd}^{initial}$ is the initial supply voltage, V_{dd}^{new} is the new supply voltage, and V_t is the threshold voltage for the chosen technology. The energy consumption is obtained using the new supply voltage. Note that, the V_{dd} scaled SIMD implementations have approximately the same performance as the non-SIMD implementations. If V_{dd} scaling is performed as described above, our techniques result in an energy savings of upto 66.26% (average of 55.88%). These benefits come at the expense of minimal area overheads (average of 2.18%).

To illustrate the optimization potential of our techniques, we performed an additional experiment with benchmark Add.bench. For this benchmark, we generated the area-performance-energy tradeoff envelope obtained using high-level synthesis with and without the use of SIMD functional units. The tradeoffs were explored by varying the resource constraints given to the high-level synthesis tool, and optimizing the performance for each resource constraint. Each RTL circuit was synthesized and technology mapped, and from the area-performance-energy estimates of each synthesized netlist, we obtained one point on the area-performance-energy curve. The performance-energy tradeoff was explored through the use of V_{dd} scaling. The area-performance-energy design envelope for the conventional and SIMD architectures is shown as a 3-dimensional surface in Figures 7(a) and 7(b), respectively. The figures clearly show that the use of SIMD units during high-level synthesis results in an expansion of the area-delay-energy envelope, to include significantly higher quality designs.

To clearly show the improvements obtained through the use of SIMD functional units, the area-performance and area-energy curves for the two architectures were obtained by projecting the maximum performance points from the 3-dimensional surfaces onto the area-energy and area-performance planes, respectively. These curves are plotted in Figures 8(a) and 8(b). The curves clearly suggest that:

- At the cost of a negligible area overhead, the performance and energy consumption of a design can be *simultaneously* improved through the use of SIMD functional units.
- The use of SIMD functional units enables a desired performance to be attained with a circuit that requires smaller area and energy consumption.

For example, consider point P in Figure 8(a), that represents a design without SIMD functional units. Through the use of SIMD func-

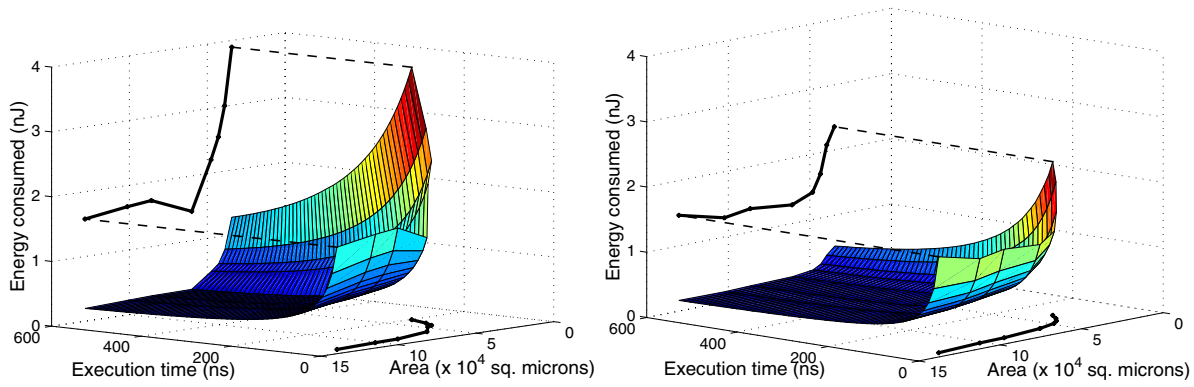


Figure 7. Area-Delay-Energy tradeoff envelope for the example benchmark Add_bench (a) without SIMD units, and (b) with SIMD units

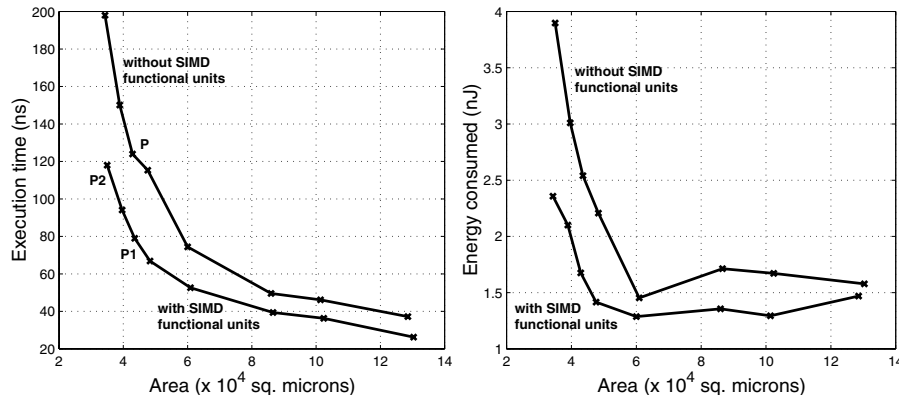


Figure 8. (a) Area-Delay tradeoff curves with and without the use of SIMD functional units for the example benchmark Add_bench (b) Area-Energy tradeoff curves for the same benchmark

tional units, we can obtain simultaneous performance improvement (of 1.57X) and energy savings (34%) with negligible area overhead (point P1). Alternatively, we can obtain a simultaneous decrease in the area (18.4%) and energy consumption (7.22%) of the design with a slight improvement in performance (point P2). Thus, we believe that appropriate use of SIMD functional units can lead a high level synthesis tool to explore portions of the design space that were not previously possible to reach, resulting in a superior area-delay-energy tradeoff space.

VI. CONCLUSIONS

We have presented a technique to integrate SIMD functional units into the scope of a general high-level synthesis system. We have presented hardware architectures for SIMD arithmetic modules, and have described how the various high-level synthesis tasks can be enhanced to translate the optimization potential of these functional units into maximal performance improvements for the entire design. In addition, the use of SIMD functional units leads to a decrease in the energy consumption of a design. The performance improvements and energy savings are simultaneous, thus improving the quality of the area-performance-energy design space. We have demonstrated through experimental results using several example benchmarks that significant performance improvements and energy savings are achieved through the use of our SIMD unit based synthesis techniques.

ACKNOWLEDGEMENTS

This paper is based in part on research funded through the DARPA PAC/C program under AFRL Contract F30602-00C-0154.

References

- [1] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture" in *IEEE Micro*, vol. 16, no. 4, pp. 42–50, 1996.
- [2] G. Goldman and P. Tirumalai, "UltraSPARC-II: the advancement of ultracomputing" in *Proc. COMPCON*, 1996.
- [3] R. B. Lee and J. Huck, "64 bit and multimedia extensions in the PA-RISC 2.0 architecture" in *Proc. COMPCON*, pp. 152–160, 1996.
- [4] "Closing the Gap Between ASIC and Custom: Design Examples", special session #27 at the *IEEE/ACM Design Automation Conf.*, June 2001.
- [5] N. Takagi, "A multiple precision modular multiplication algorithm with triangle additions" in *IEICE Trans. on Info. and Systems*, vol. E78, 1995.
- [6] M. E. Louie and M. D. Ercegovac, "A variable precision square-root implementation for field programmable gate arrays" in *Journal of Supercomputing*, vol. 9, no. 3, pp. 315–336, 1995.
- [7] C. G. Zhou *et al.*, "MPEG video decoding with the UltraSPARC Visual Instruction Set" in *Proc. COMPCON*, 1995.
- [8] J. F. Blinn, "Fugue for MMX" in *IEEE Computer Graphics and Applications*, vol. 17, no. 2, pp. 88–93, 1997.
- [9] Z. J. A. Mou, D. S. Rice, and D. Wei, "VIS based native video processing on UltraSPARC-II" in *Proc. Intl. Conf. Image Proc.*, pp. 153–156, 1996.
- [10] D. D. Gajski, N. D. Dutt, A. Lu, and S. Lin, *High Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, Norwell, MA, 1992.
- [11] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.
- [12] M. D. Ercegovac, D. Kirovski, G. Mustafa, and M. Potkonjak, "Behavioral synthesis optimization using multiple precision arithmetic" in *Proc. ICASSP*, pp. 3113–3116, 1998.
- [13] M. D. Ercegovac, D. Kirovski, and M. Potkonjak, "Low-power behavioral synthesis optimization using multiple precision arithmetic" in *Proc. IEEE/ACM Design Automation Conf.*, pp. 568–573, 1999.
- [14] V. Agarwal, A. Pande, and M. M. Mehendale, "High level synthesis of multi-precision data flow graphs" in *Proc. IEEE Intl. Conf. on VLSI Design*, pp. 411–416, January 2001.
- [15] "Mosis", (<http://www.mosis.org/Technical/Processes/menu-processes.html>).
- [16] "DC Users Manual", Synopsys Inc., (<http://www.synopsys.com>).
- [17] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proc. IEEE/ACM Design Automation Conf.*, pp. 491–496, 1994.