FleXilicon Architecture and Its VLSI Implementation

Jong-Suk Lee and Dong Sam Ha, *Fellow, IEEE*

***Abstract*—In this paper, we present a new coarse-grained reconfigurable architecture called FleXilicon for multimedia and wireless communications, which improves resource utilization and achieves a high degree of loop level parallelism (LLP). The proposed architecture mitigates major shortcomings with existing architectures through wider memory bandwidth, reconfigurable controller, and flexible word-length support. VLSI implementation of FleXilicon indicates that the proposed pipeline architecture can achieve a high speed operation up to 1 GHz using 65-nm SOI CMOS process with moderate silicon area. To estimate the performance of FleXilicon, we modeled the processor in SystemC and implemented five different types of applications commonly used in wireless communications and multimedia applications and compared its performance with an ARM processor and a TI digital signal processor. The simulation results indicate that FleXilicon reduces the number of clock cycles and increases the speed for all five applications. The reduction and speedup ratios are as large as two orders of magnitude for some applications.**

***Index Terms*—Array processing, loop-level parallelism, reconfigurable architecture, system-on-chip (SOC).**

I. Introduction

MULTIMEDIA and wireless communication applications demand high computing power, flexibility, and scalability. An application-specific integrated circuit (ASIC) solution would meet the high computing power requirement, but is inflexible and not scalable. On the other hand, general purpose microprocessors or digital signal processing (DSP) chips are flexible, but often fail to provide sufficient computing power. Various processor architectures such as VLIW or vector processors have been introduced to increase the computing power, but the computing power is still insufficient or the architectures are too power hungry or expensive in silicon. Since early 1990's, reconfigurable architectures have been proposed as a compromise between the two extreme solutions, and been applied for multimedia and wireless communication applications as surveyed in [1] and [2]. Reconfigurable architectures are flexible and scalable and can provide reasonably high computing power, and hence they are suitable for multimedia and wireless communication applications.

A reconfigurable architecture has evolved from the logic-level fabric to the processing-level fabric [1], [2]. The logic level fabric is a fine grained architecture, in which logic level circuits are mapped into configurable lookup tables (LUTs) and routing. In contrast, the processing level fabric is a coarse grained architecture, which incorporates predesigned processing elements such as adders, multipliers, shifters, and logical units as building blocks. The processing-level fabric has several advantages over the logic-level fabric such as efficient area, high performance, and low power, but it suffers from low flexibility and inefficiency at bit-level operations [2], [3]. This paper concerns only coarse grained architectures due to those reasons.

Some critical loop operations such as discrete cosine transform and motion estimation for multimedia applications and filter operations, equalization operations in wireless communication applications usually consume a good portion of the total execution cycles. The key issue in implementing multimedia or wireless algorithms onto a reconfigurable architecture is to map critical loops into processing elements optimally to meet the computing need. Two major techniques for efficient execution of loops for reconfigurable architectures are pipelining and loop level parallelism (LLP). The pipelining technique, which is widely employed for coarse-grained reconfigurable architecture, achieves high throughput. Several compilers are available to generate a pipelined datapath from a given data flow graph and to map the pipelined datapath onto processing elements [4]–[9]. The LLP technique was investigated initially for parallel computing machines such as supercomputers and multiprocessor systems, and it executes multiple iterations concurrently in a loop with multiple processors [10]–[12]. The LLP can be a good technique for mapping a loop into a reconfigurable architecture, since it achieves a significant speedup with a large number of processing elements.

While various types of classifications for coarse grained architectures were made in previous papers [1], [2], [13], we categorize existing coarse grained architectures into two groups, datapath-oriented and instruction-oriented, based on the type of instructions performed by underlying processing elements. A processing element for a datapath-oriented architecture executes only one type of operation once it is configured, and a required dataflow is constructed by routing necessary processing elements.

A datapath-oriented architecture usually has mesh-structured processing elements, and the architecture is suitable for mapping loops into a pipelined datapath, which achieves high throughput. However, in general, the architecture results in low resource utilization. Several existing architectures such as MATRIX [14], REMARC [15], MorphoSys [16], and PactXPP [17] belong to this group. To implement the LLP on a datapath-oriented architecture, the body of the loop is replicated on a mesh, and multiple iterations are executed concurrently using hybrid of both pipelining and LLP techniques. This scheme is employed for Chameleon architecture presented in [22]. However, low resource utilization still remains as problematic because of large redundancy introduced during the mapping.

In contrast, a processing element of an instruction-oriented architecture performs a sequence of operations, which are de-

Manuscript received August 25, 2007; revised March 09, 2008. First published May 02, 2009; current version published July 22, 2009.

The authors are with Virginia Polytechnic Institute and State University, Blacksburg, VA 24130 USA (e-mail: watsup@vt.edu; ha@vt.edu).

Digital Object Identifier 10.1109/TVLSI.2009.2017440

1063-8210/$26.00 © 2009 IEEE

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

fined by instructions, microcodes, and/or control signals. Instructions are stored in a configuration memory and fetched by a controller to control the processing element. As a processing element can execute the entire body of a loop, the LLP is simply to assign multiple processing elements running concurrently. Existing reconfigurable architectures belong to this group include RAW [18], PADDI [19], Chameleon [20], and AVISPA [21]. We consider an instruction-oriented architecture in this paper due to high resource utilization.

Although instruction-oriented architectures offer higher resource utilization compared with data-oriented architectures, there are three major shortcomings for existing reconfigurable machines. First, since the LLP increases simultaneous memory access linearly to the number of parallel operations, existing machines suffer from shortage of available memory bandwidth. It is usually the limiting factor for high performance. Second, since the reconfiguration capacity of a controller should be set for the largest loop body, it results in large overhead and performance degradation for existing architectures. Finally, the number of processing elements should be sufficiently large to achieve a high degree of parallelism in the LLP. To mitigate the problems, we propose a new instruction-oriented reconfigurable architecture called FleXilicon [23]. FleXilicon increases the memory bandwidth with employment of a crossbar switch network (XBSN). FleXilicon adopts a reconfigurable controller, which reduces the overhead associated with execution of instructions. In addition, flexible word-length operations for FleXilicon increase the sub-word parallelism (SWP) [24].

This paper is organized as follows. Section II describes a resource utilization issue in the LLP and shortcomings with existing architectures. Section III presents the proposed reconfigurable architecture and explains how the shortcomings are addressed in the proposed architecture. We also describe VLSI implementation of the architecture. Section IV presents simulation results and compares the performance of the proposed architecture with conventional processors. Section V draws some conclusions on the proposed architecture. Finally, it should be noted that this paper is an extension of an earlier and brief version presented in [23].

II. Preliminary

In this section, we discuss resource utilization of existing architectures and critical design issues considered in developing our architecture.

A. Resource Utilization

Resource utilization is a key factor to achieve high performance for reconfigurable architectures, and it can serve as a key metric to decide an appropriate architecture type. As categorized earlier, two different types of existing architectures such that datapath-oriented and instruction-oriented architectures have different resource utilization depending on the mechanism to execute loops. Consider an example loop in Fig. 1. Fig. 1(a) shows an example pseudo code for a simple N iterative loop. The loop body of the code can be transformed to five operations as shown in Fig. 1(b). Fig. 1(c) shows a transformed data flow graph (DFG) of the loop body, which can be mapped to processing elements.

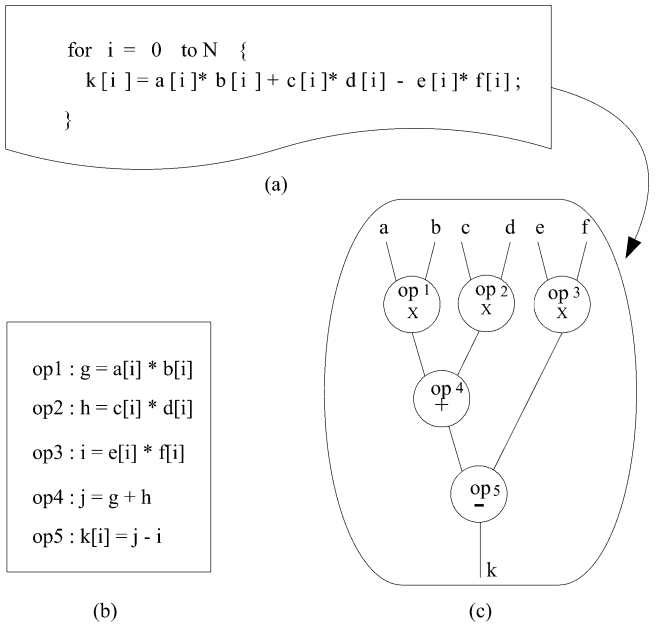

Fig. 1. Example loop and a data flow graph of its loop body. (a) Pseudo code of the N iterative loop. (b) Operation assignment results. (c) DFG of the loop body.

Fig. 2 shows a mapping of the DFG in Fig. 1 onto an 8×8 mesh datapath oriented architecture, which employs pipelining and the LLP. In this mapping, we assume that a processing element (PE) has four possible connections with its neighbor PEs. Each PE is configured as a required operator, and interconnections are configured to form the datapath flow. Note that delay elements are necessary for the pipelining. Fig. 2(a) indicates the case where input/outputs (I/Os) are available only at the boundary of the mesh. Most PEs are idle during the operation, and some of them are assigned simply as delay elements to provide interconnections to the I/O bus. Note that PEs usually do not have large memory to store temporary results of loop calculations. Further, only two iterations can be mapped onto the mesh due to lack of available input ports, and hence the degree of the LLP is two. Only 10 PEs out of 64 PEs are used for actual processing to result in 15.6% of resource utilization. When an unlimited number of I/Os are available as shown in Fig. 2(b), the degree of parallelism increases to eight, and the resource utilization to 62.5%. Note that an unlimited I/O accessibility alone does not guarantee high resource utilization because of the mismatches between a mesh structure and the DFG of a loop body. In addition, severe reconfiguration overhead incurs if the size of a DFG exceeds the size of a given mesh. For some applications such as infinite-impulse response (IIR)/finite-impulse response (FIR) filters, the datapath oriented architecture can be a better choice provided a DFG matches with the mesh structure.

Fig. 3 illustrates the LLP for execution of loops on an instruction oriented architecture with unlimited I/Os, in which operations of a loop are executed on the same PE sequentially, and multiple iterations are executed concurrently on different PEs. In contrast to datapath oriented architectures, high resource utilization is achieved for instruction oriented architectures as long as the number of iterations of a loop exceeds the number of available PEs, and it is usually the case for our target applica-

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

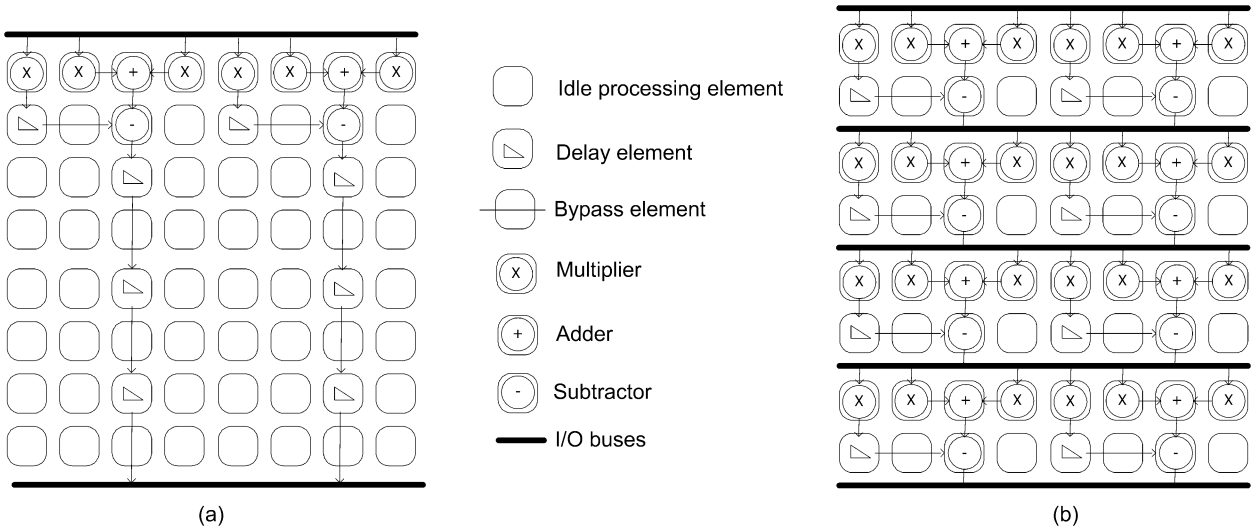

Fig. 2. Mapping onto a datapath oriented architecture (a) with limited I/Os and (b) with unlimited I/Os.

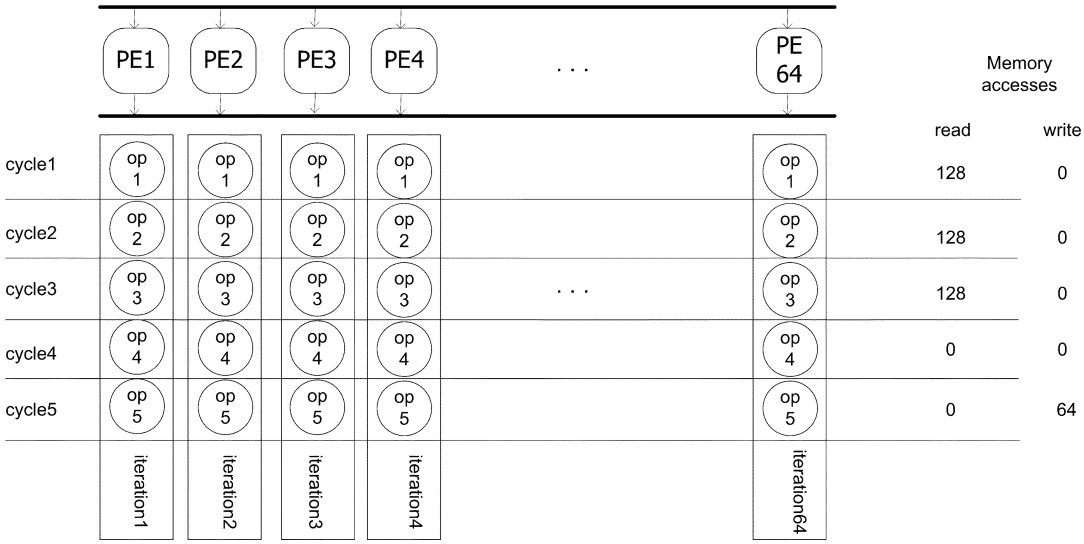

Fig. 3. LLP for an instruction oriented architecture with unlimited I/Os.

tions. Therefore, instruction oriented architectures offer higher resource utilization and hence better performance than datapath oriented architectures. However, existing instruction oriented architectures does not exploit the full potential of the LLP due to several limitations as described in the following subsection.

B. Design Issues

There are three major design issues, memory bandwidth, controller design, and sub-word parallelism, in the LLP for instruction oriented architectures.

1) Memory Bandwidth: An instruction oriented architecture has a higher demand on simultaneous peak memory access than a datapath oriented architecture. For examples, the instruction oriented architecture in Fig. 3 requires 128 input and 64 output memory accesses at certain peak times, while the datapath oriented architecture in Fig. 2 requires 48 input and 8 output memory accesses constantly. PEs with a limited memory bandwidth should wait at certain peak cycles until necessary data is available from the memory. Therefore, wide memory bandwidth is a critical design issue to realize high degree parallelism for the LLP. However, existing instruction oriented architectures fail to provide sufficient memory bandwidth to maximize the LLP. To name a few, Chameleon [20] provides configurable memory access up to 128 bits, which is insufficient for seven 32-bit PEs. PADDI [19] uses a crossbar switch to provide non-conflict connections among PEs, but it has a limited memory access capability. The memory system for the proposed architecture provides guaranteed operand access from local memories to PEs, which maximizes the LLP.

2) Controller Design: A memory-based controller determines the operations of PEs for a conventional instruction oriented architecture. A sequencer generates global instructions, which, in turn, select VLIW-like instructions of a reconfigured memory. These memory based controllers have several problems as described next. First, the size of an instruction memory is typically small such as eight entries for Chameleon [20] and PADDI [19]. If single iteration requires a larger number of instructions than supported by the instruction memory, the

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

instruction memory should be reconfigured. It causes a severe degradation of the performance. To support a larger number of instructions, the memory size should be increased, which results in large area overhead and configuration time overhead of the controller. Second, to share a localized controller among PEs, it needs multiplexers and demultiplexers, which complicates the design as well as large silicon area, hence independent controllers are required even when all PEs have the same functionality. Third, a memory-based controller is not suitable for control of instruction pipelines, as each pipeline stage requires different memory locations. Hence, it necessitates a large size memory for super-pipeline stages. Finally, to process branches or control flows, a sequencer should operate at a higher frequency than PEs, which might limit the operating frequency of the PEs.

PADDI [19] uses an external sequencer as a global controller which generates 3-bit global instructions. The global instruction points eight different nano-store memories, which contain eight 53-bit VLIW instructions. Similarly, Chameleon [20] has a control logic unit (CLU), which consists of a PLA for finite state machine and selects eight-word instruction memory to control datapath units (DPUs). AVISPA [21] has VLIW-like controllers and a configuration memory. Unlike other instruction oriented architectures, RAW [18] uses a microprocessor as a PE. Hence, instructions are fetched and decoded to execute operations like a conventional microprocessor. As the result, the area overhead for instruction cache, instruction fetch logic and decoder logic is high.

3) Sub-Word Parallelism: Algorithms in multimedia and wireless communication applications require various precisions of data. For example, audio algorithms generally require high precision ranging from 16 bits to 24 bits. An 8-bit to 16-bit resolution is common for video algorithms. A wide range of precisions from 4 to 32 bits are used for wireless communication algorithms. Sub-word parallelism (SWP) is a method to increase the parallelism by partitioning a datapath into sub-words, so that multiple sub-word data can be processed concurrently [24]. Therefore, the SWP can be used effectively for parallel processing of the various precision data in multimedia and wireless communication applications.

Only a few of reconfigurable architectures adopt SWP in a limited manner. PADDI [19] supports 32-bit additions by concatenating two 16-bit execution units (EXUs). Chameleon [20] supports two 16-bit additions and single 32-bit additions in DPU. In addition, Chameleon provides two types of multiplications, 16×24 and 16×16, without the benefit of increased parallelism. Note that none of mesh structured architectures supports the SWP since additional interconnections among PEs is very costly.

III. Proposed Reconfigurable Architecture

Our proposed reconfigurable architecture FleXilicon intends to provide massive parallel processing of loops for multimedia and wireless communication applications. FleXilicon is intended for use as a coprocessor attached to a host processor, as it is optimized for data dominated loop operations rather than control dominated operations (which are suitable for a general purposed processor). Typically, FleXilicon performs only critical loop operations, while the host processor handles remaining tasks. Also, the host processor is responsible for managing FleXilicon through its control registers. In this section, we describe the overall architecture of FleXilicon and then our schemes to address shortcomings of exiting instruction oriented architectures.

A. Overall Architecture

FleXilicon has an array of n processing element slices (PESs), where n is scalable. A PES is the basic block for the LLP and consists of an array of processing elements, which enables execution of multiple iterations of a loop in parallel. It is also feasible to allocate different outer loops or simultaneous multi-threads to different multiple PESs. A PES is connected to only its neighbor PESs for easy scalability and to a host processor through a high-speed system bus.

Fig. 4 shows the overall architecture of FleXilicon. One PES consists of two local memories, an XBSN (Crossbar Switch Network), 16 processing element and multipliers (PEMs), and a reconfigurable controller (RC). The number of operations that can be executed simultaneously on 16 PEMs depends on the type of the operation such as 32 8-bit arithmetic logic unit (ALU) operations and 16 8×8 multiplications. The local memories provide storages for I/O data streams to be read/written by the host processor and for temporary data for the PES and other neighboring PESs. The XBSN provides various types of memory accesses and flexible word length operations. The reconfigurable controller is responsible for generating control signals for the local memories, the XBSN, and the 16 PEMs.

One PEM can perform single 8×8 multiply and accumulate (MAC) operation and two 8-bit ALU operations, and it consists of two PEs, two partial accumulators (PACCs), and one 9×9 multiplier. A PE consists of three 8-bit ALUs, five 8-bit data registers, a status register, and a carry controller. The state register stores four different kinds of states—negative, overflow, zero, and carry according to the execution result of ALUs. The values of the status register are transmitted to the reconfigurable controller, so that the controller switches the state of the state machine based on the execution results. To support multiple word operations, the carry controller propagates the carry from the previous stage. To protect results from overflows or underflows during accumulations, two PACCs may be configured as a 16-bit accumulator for iterative single 16-bit accumulations or two 8-bit accumulators for two independent 8-bit PE operations. A PE supports various operations including general ALU operations such as addition, subtraction, logic operation, and configurable application specific operations such as add compare select (ACS), sum of absolute difference (SAD), weighted sum, and clipping operation. Other application specific operations may be added to a PE by configuring the datapath of the PE. These configurable operations reduce the number of clock cycles for loop processing when implementing algorithms of wireless communications and multimedia applications.

B. Processing Element Slice (PES)

As noted above, a PES is the basic processing unit for the LLP, and its structure is shown in Fig. 5. To provide enough memory bandwidth for the LLP, a PES has two 16 kB (512 entries with

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

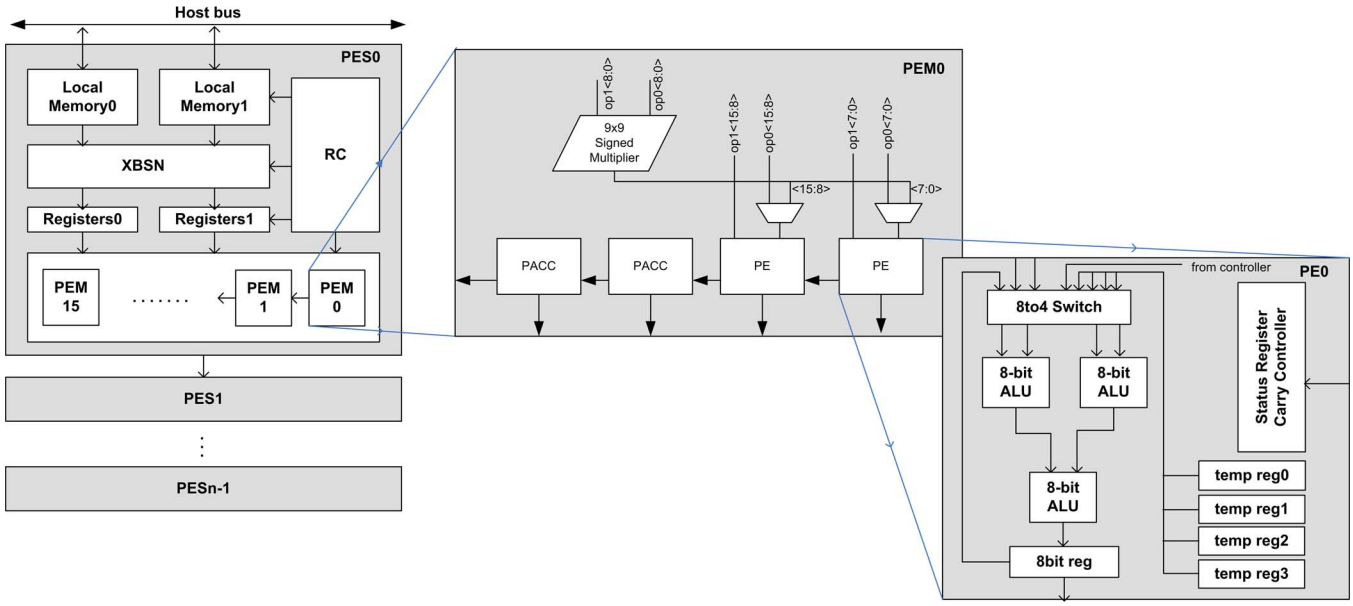

Fig. 4. Overall architecture of FleXilicon.

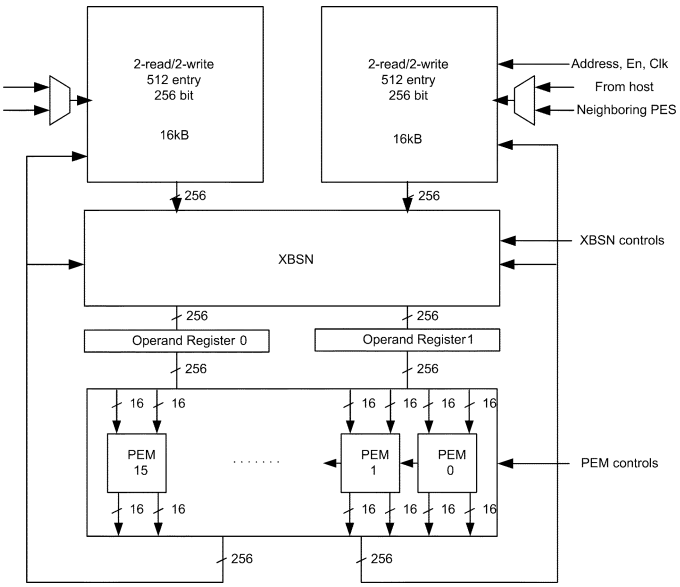

Fig. 5. Structure of a PES.

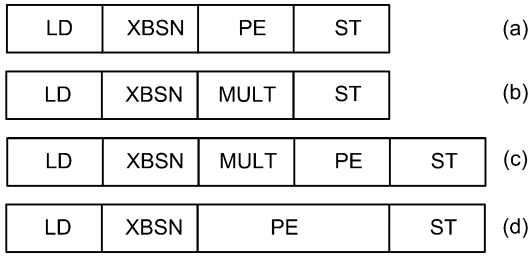

Fig. 6. Instruction pipeline stages. (a) PE operation. (b) Multiplication operation. (c) MAC operation. (d) Multi-cycle PE operation.

256-bit each) memories. Two independent addresses index two 256-bit data of the two local memories. Through the XBSN, two operand registers fetch 512-bit operand data to be executed by 16 PEMs. The XBSN includes two 32×32 8-bit crossbar switches, so any 8-bit word among 32 operands can be fetched to any operand register.

A local memory has 256-bit wide dual I/O ports (two ports for read and two ports for write), which enable simultaneous read/write accesses for the host processor and for PEMs within the PES and/or those for neighboring PESs. The memory system enables a fetch of 64 8-bit operand data in single clock cycle, equivalently, two operand data for each PE, under any operating condition. Hence, it meets the peak demand for memory access during loop executions. Since the versatility of memory access is an important factor to support various types of algorithms in multimedia and wireless communication applications and to minimize the communication overhead among PEs, the XBSN provides various types of memory accesses including broadcasting and butterfly access. Versatile memory accesses enable efficient execution of various algorithms which requires complex memory accesses during loop executions.

There are two writing channels for local memories. The first one is for the external memory access, which is multiplexed with a neighboring PES and a host processor. The second one is a dedicated channel for local PEMs. 16 PEMs generate 64 bytes (4 bytes for each PEM including 2 byte accumulation data), and they are transferred to either local memories directly without data alignment or operand registers with data alignment through XBSN.

Fig. 6 shows a few configurable instruction pipeline stages for a PES. LD is the first pipeline stage, where 512-bit data are loaded from the local memory to sense amplifiers. XBSN is the operand fetch stage through the XBSN, PE is the execution stage, MULT is the multiplication stage, and ST is memory write stage. Fig. 6(d) is the case when multiple clock cycles are required for the execution stage.

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

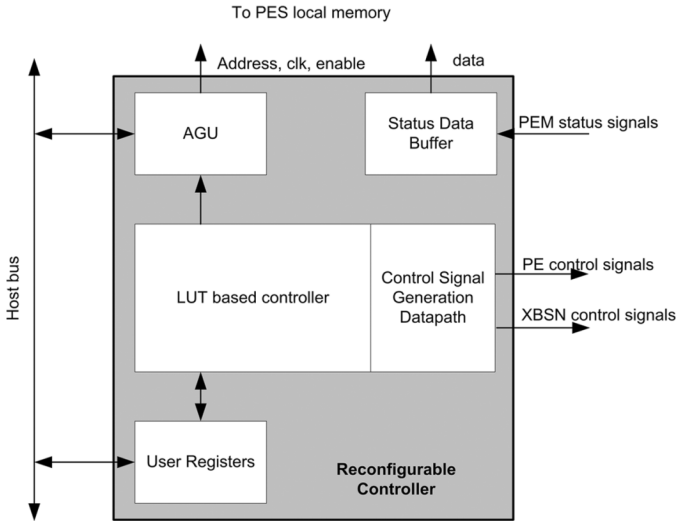

Fig. 7. Structure of a controller.

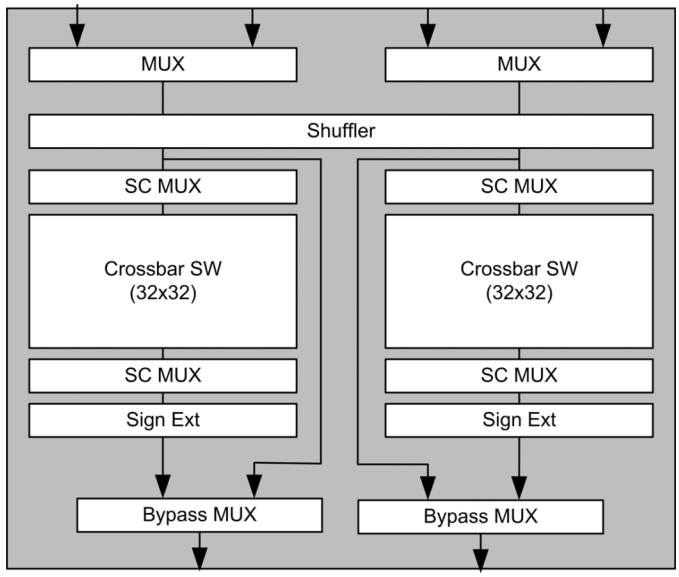

Fig. 8. Structure of an XBSN.

C. Reconfigurable Controller

The controller of a PES generates appropriate control signals for instruction pipeline stages—LD, XBSN, MULT, PE, and ST. Fig. 7 shows the structure of an LUT-based controller, which consists of LUTs, user programmable registers (called user registers), an address generation unit (AGU), a control signal generation datapath, and a status data buffer. The host processor communicates with the controller through user registers to initiate the controller or to retrieve the status of the controller and the PES. The status data buffer stores the values of the status registers in PEs, which can be transferred to the local memories.

To mitigate the shortcomings of existing memory-based controllers, the proposed controller adopts a fine-grained reconfiguration like an FPGA. Unlike previous memory-based controllers, the proposed controller maps combinational logic onto LUTs and employs high speed predesigned datapaths to generate control signals. The complexity of the proposed controller is, in general, simpler than existing memory-based ones, since it supports only required operations of the loop, not all possible functionality of a PE. Further, while mapping the controller onto LUTs, conventional logic optimization techniques can be applied for logic minimization. Additionally, a controller can be shared across multiple PESs to exploit the benefit of the LLP.

The current configuration of controller has 128 CLBs with 1024 LUTs and 1024 F/Fs, which is equivalent to a Xilinx Virtex2 xc2v80 [37]. The configuration stream is transferred through the system bus from the host. The maximum size of the configuration bits for the controller including LUTs, and the mapping memory is 73.64 kB, and the reconfiguration time is around 37.7 us assuming the system bus speed of 1 GB/s (32-bit at 250 MHz).

The AGU, consisting of four address registers and three address generation adders, is a predesigned block responsible for generation of control signals for local memory access. The control signal generation datapath provides generation and remapping of control signals for the XBSN and the PEMs. A hybrid of a datapath scheme and a mapping memory scheme are adopted for an XBSN block, since it requires complicated calculations for generating control signals (such as for shift and rotate operations) and needs to store frequently used configurations in the mapping memory. Some PEM control signals are generated from the datapath, while others are simply fetched from the mapping memory or bypassed from the LUT. Our sophisticated controller, when compared to existing memory based controllers which simply store configurations of datapaths, leads to more versatile functionality and potentially higher speed operations for the same controller area.

D. SWP With Flexible Word-Length Support

FleXilicon embedding the proposed XBSN supports flexible word-length operations such as shift, MAC, and addition/subtraction, which leads to a high degree of SWP. The proposed XBSN as shown in Fig. 8 consists of multiple sets of multiplexers, which provide various dataflow paths. The XBSN supports multiple various word length shift operations, specifically 8-, 16-, and 32-bit shifts. Fig. 9 illustrates one-bit arithmetic shift right operations using scrambling multiplexers (SC_MUXs) and a 32×32 8-bit crossbar switch. An SC_MUX performs a bit scrambling operation, which produces 8-bit words by collecting corresponding bits from the operands. For example, all LSBs of the operands $(a0, b0, c0, d0, e0, f0, g0)$ are packed into a single word. The XBSN performs a necessary shift or rotate operation, and the result is passed to the descrambling multiplexer SC_MUX. The scrambled data are descrambled to restore the original bit ordering. Scramble and descramble multiplexers with incorporation of a crossbar switch eliminate the need for multiple barrel shifters to save the area. It also provides multiple flexible word-length shift operations, which is not feasible with a conventional barrel shifter.

Various reconfigurable multiplication and MAC architectures have been published [38], [39]. We propose a new scheme called

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

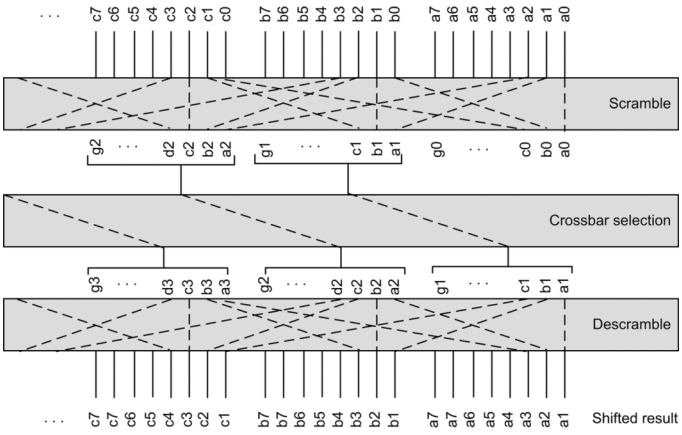

Fig. 9. One-bit arithmetic shift right operation.

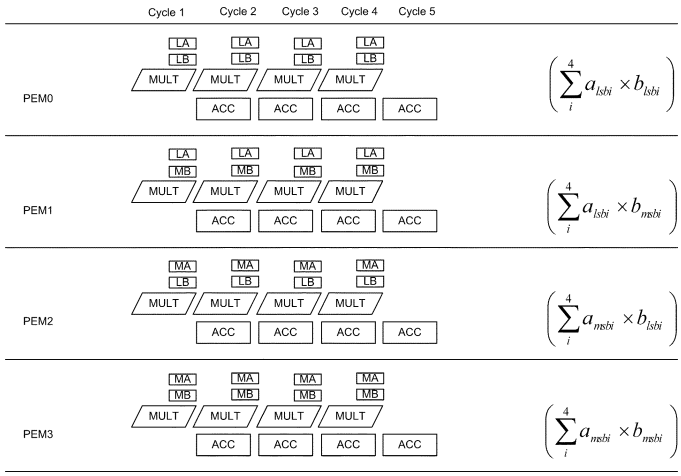

Fig. 10. Pipelined execution of the DASM.

DASM (Divide, Accumulate and Sum Multiplication) to provide flexible word-length MAC operations targeting for the instruction oriented architecture. As described in [40], a MAC operation can be expressed with lower precision multiplications as in (1). A_i and B_i are the multiplicand and the multiplier, respectively, where i is the accumulation index. The scaling factor is used to align the partially accumulated results for the summation. A partial multiplication can be expanded repeatedly with half precision multiplications until it reaches atomic multiplications. Partitioned multiplication results are partially accumulated first, and scaled partial products are summed later. This method enables high speed flexible word-length MAC operations, which can be accomplished with parallel 9-bit multipliers (implemented with PEMs). Even though additional sum cycles are required for the summation of partial products, MAC operations are accelerated with parallelized high speed multiplications, and summation of the partial products are required only at the end on MAC iteration

$$\begin{aligned}\sum_i A_i \times B_i = &\left(\sum_i a_{\mathrm{msb}i} \times b_{\mathrm{msb}i}\right) \cdot 2^{w} \\ &+ \left(\sum_i a_{\mathrm{msb}i} \times b_{\mathrm{lsb}i}\right) \cdot 2^{\frac{w}{2}} \\ &+ \left(\sum_i a_{\mathrm{lsb}i} \times b_{\mathrm{msb}i}\right) \cdot 2^{\frac{w}{2}} \\ &+ \left(\sum_i a_{\mathrm{lsb}i} \times b_{\mathrm{lsb}i}\right) \qquad (1)\end{aligned}$$

where a_{msb} is the most significant part of A, a_{lsb} is the least significant part of A, w is the bit width of A and B.

Fig. 10 illustrates the DASM procedure for 16×16 MAC operations. Four iterative 16×16 MAC operations can be performed with four 8×8 PEM units as shown in Fig. 10. An XBSN divides multiplicands into 8-bit chunks and feeds them (with the sign extension) to appropriate PEM units. Each PEM accumulates four independent and divided MAC operations in five cycles, and four partial results are summed in four cycles including the XBSN pipeline cycles. FleXilicon provides various types of single or multiple MAC operations (such as sixteen 8×8, eight 16×8, five 24×8, four 32×8, four 16×16, two 24×16, two 32×16, and one 32×32 MAC) using 8×8 atomic MAC units of PEMs. An XBSN supports various sizes of multiplicands necessary for flexible word-length MAC operations.

Finally, 8-bit PEs in multiple PEMs can be concatenated to construct a wide bit-width adder or subtractor, in which each PE is configured as a carry selection adder to minimize the critical path delay.

E. VLSI Implementations and Their Performance

FleXilicon with a single PES configuration was implemented with a mix of standard cell based design and custom circuit design in 65-nm CMOS SOI process technology. Local memories and high-speed data paths were implemented as custom circuits, while the remaining blocks were implemented using standard cells. A 64×64 bit multi-port RAM macro was developed to construct local memory blocks. To provide low latency, a local memory was designed for single cycle memory access. An XBSN was implemented with 2×1 and 4×1 multiplexers composed of transmission gates.

Unlike conventional architectures with 16 or 32 bits as the size for atomic operations, 8 bits are the atomic operation for FleXilicon. Hence, high speed 8-bit adders are essential to achieve high performance for FleXilicon. Existing high speed adders focus on wide bit-width additions such as Breton Kung (BK) [25], carry look-ahead adder (CLA) [26], bypass adder [26], parallel prefix (PPrefix) [27], whose speed improvement are rather insignificant for 8-bit additions. We investigated speed optimization for 8-bit adders, which is based on a 1-bit bypass scheme. Note that conventional bypass adders typically bypass 4 bits, which are slow for 8-bit additions. Fig. 11 shows the proposed 1-bit bypass scheme for 8-bit adders. The signal $\mathrm{pp}i$, where $i = 0, 1, \ldots, 6$, denotes that the carry propagates from the previous bit stage. The signal $\mathrm{c}i$ is the carry signal from the previous bit i, and the signal $\mathrm{pc}i$ is the carry signal from the stage $i-1$. The proposed adder is similar to a Manchester carry-chain adder [26], besides it has a bypass path. Unlike conventional bypass adders in which carry paths are isolated from bypass paths, a carry and a bypass for the proposed adder share the same path through incorporation of transmission

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

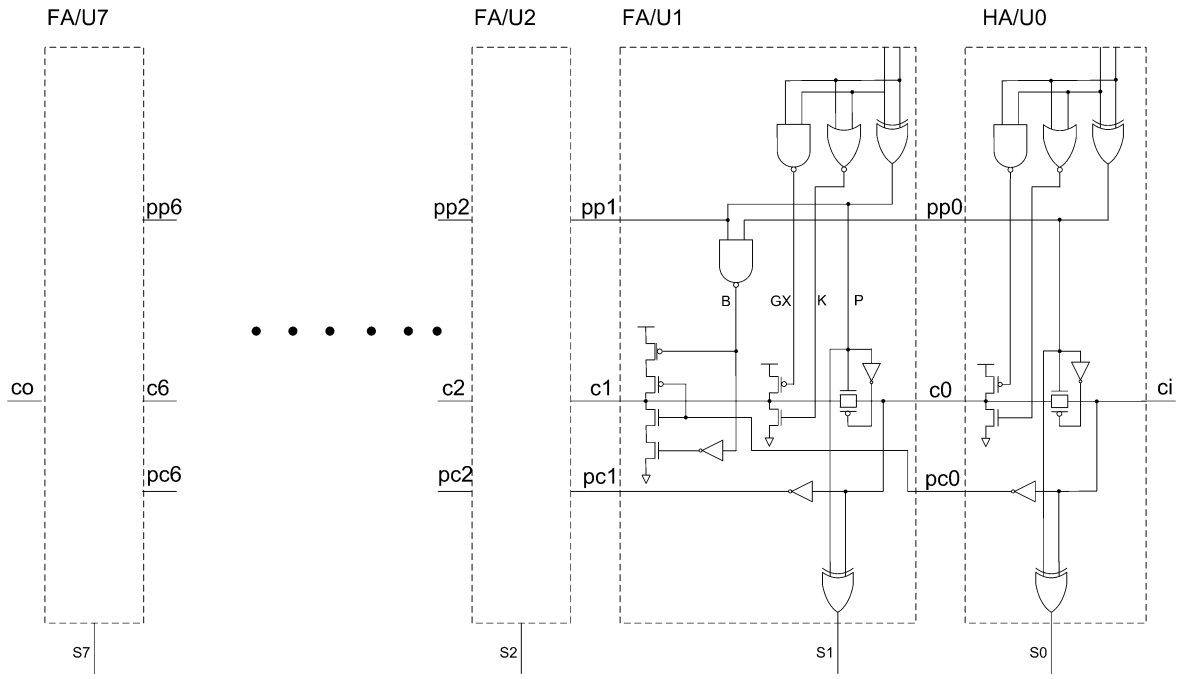

Fig. 11. Proposed 1-bit bypass 8-bit adder.

TABLE I
CRITICAL PATH DELAY OF DIFFERENT TYPES OF 8-BIT ADDERS

	CRA	*CLA*	*BK*	*Pprefix*	*Proposed*
Delay	172ps	170ps	167ps	188ps	84ps
Speedup	2.05x	2.02x	1.99x	2.24x	-

gates. In other words, both $\mathrm{p}ci$ and ci signals from the previous stage drive the shared carry path of a stage. This scheme works as the bypass signal, if arrives earlier than the carry, triggers the shared carry path to accelerate the carry propagation on the critical carry path.

For the comparison with previous adders, we synthesized the BK, PPrefix, and CLA using Synopsys DesignWare library [28] and implemented a carry ripple adder (CRA) at a schematic level using a standard cell library. Critical path delays of synthesized adders were obtained through static timing analysis using Synopsys Primetime [29]. The delay of the proposed adder was obtained through SPICE simulation. To calibrate static timing analysis results with SPICE simulation results, the delay of the CRA was simulated in both methods and compared the results each other to obtain a calibration factor.

Table I shows critical path delays of various types of adders for a CMOS SOI 65-nm technology under the supply voltage of 1.3 V. The proposed adder has delay of only 84 ps, while all other types of adders including a CRA have delays in the range of 167 to 188 ps. The speedup of the proposed adder over other types is about two times, which is crucial for high performance of FleXilicon.

To improve the area efficiency and to optimize the routing, key blocks of FleXilicon were placed manually. For other synthesized blocks such as multipliers, Synopsys Physical Compiler was used for auto-placement and area/speed optimization, Nanorouter of Cadence was for routing and an in-house tool of a large semiconductor company was for power estimation. Single cycle 9-bit Wallace tree multipliers were synthesized using DesignWare library [28]. Fig. 12 shows the result of a cell placement of a PES. The size of single PES is 2296 μm $\times$ 1880 μm, which is reasonable for constructing an array of PESs or integration of it as a coprocessor in SOC. We noticed that two local memories (512×256 bits for each memory) occupy 74.6% of the total area, 11.12% for 16 PEMs, and 5.9% for an XBSN.

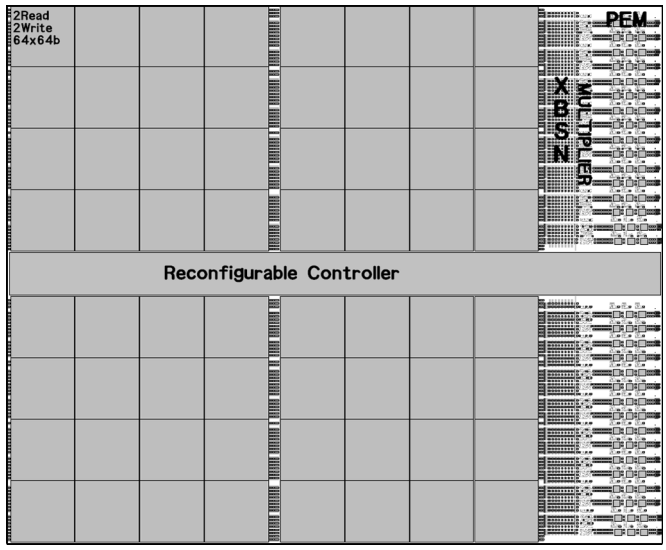

Fig. 12. Cell placement of a PES.

Fig. 13 presents static timing analysis results for each pipeline stage obtained using Synopsys PrimeTime [29]. "LD/ST" (Load/Store) is the memory access time for the LD/ST pipeline stage, "XBSN" worst case delay for the XBSN pipeline stage, "mult8" for 8-bit multiplications, "worst PE" the worst case delay for the 8-bit configurable datapath of a

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

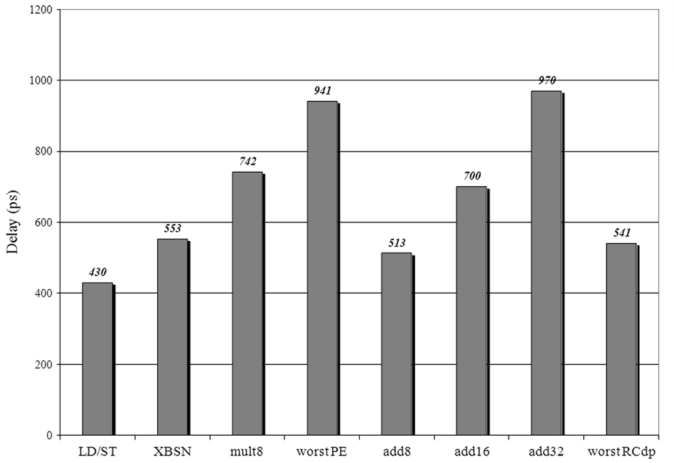

Fig. 13. Static timing analysis of a PES.

TABLE II
COMPARISON OF POWER CONSUMPTION

	Op. Freq.	*Power (mW)*	*mW/ MHz*	*MIPS/ mW*	*Area (mm^2)*
FleXilicon	1 GHz	389.35	0.39	41.10	4.32
ARM 920T	200 MHz	50.00	0.25	4.40	4.70
ARM 1176	620 MHz	279.00	0.45	3.11	*na*
TMS320C64x	1.2GHz	1,760+	1.46	5.45	*na*

+ Core, L1 Cache only

PE, and "worst RCdp" the worst case delay for the reconfigurable controller datapath. Since FleXilicon can be configured for a different word-length, we considered additions in three different word-lengths—8 bits, 16 bits and 32 bits. Our results indicate that multiplications and ALU operations can reach up to 1 GHz of the clock speed, and some other operations such as 16-bit MAC operations, 16-bit additions and 8-bit additions can be performed at an even higher clock frequency.

We estimated the power consumption of FleXilicon with a *single* PES under the supply voltage of 1.3 V, the temperature of 100 °C and the operating frequency of 1 GHz. Our simulation results indicate FleXilicon consumes total 389.35 mW, in which 16 PEMs consume 180.32 mW, an XBSN consumes 192.66 mW, and a local memory 16.37 mW. Compared to TI DSP TMS320C64x and ARM processors as shown in Table II, FleXilicon provides better power-performance efficiency (MIPS/mW). However, since ARM 920T uses 130 nm, TI DSP and ARM 1176 use 90 nm, and FleXilicon 65-nm technology, a direct comparison is unfair and not meaningful. The above comparisons may suggest only that FleXilicon's power consumption falls into a practical range.

IV. PERFORMANCE OF FLEXILICON

We modeled FleXilicon with a *single* PES in SystemC and compared its performance with an ARM processor and a TI DSP processor for several multimedia and wireless applications.

A. Simulation Environment

A system model for FleXilicon embedded in a SOC with a host processor was developed using SystemC [30]. The System-C offers an efficient modeling method for both software and hardware, and is suitable for modeling a SOC embedding FleXilicon, which is modeled as a hardware block. Application functions on the host processor were described in a C language. A reconfigurable controller was implemented as a FSM (Finite State Machine) using a SystemC hardware modeling feature, which is mapped onto LUTs using an FPGA mapping tool. The compiled SystemC model provides simulation platform and infrastructure for verification, in which a golden model of applications was compared against our implementation model.

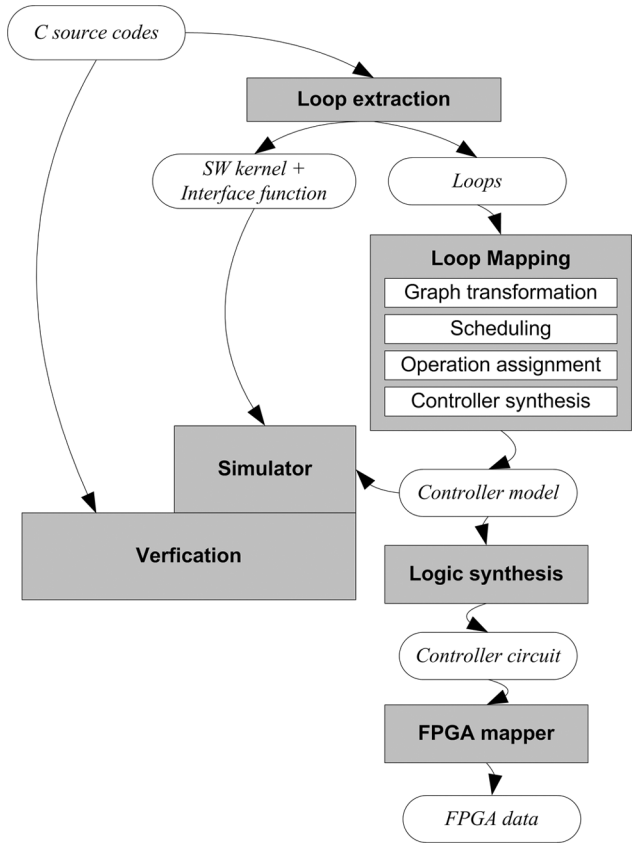

Fig. 14. Loop mapping flow.

Several steps are needed for efficient mapping of loops onto FleXilicon, while exploiting the full capability of the FleXilicon architecture, and the overall procedure is shown in Fig. 14. The loop extraction step extracts loop bodies (to be executed on FleXilicon) from C code of target applications. A loop body is selected carefully considering the task size and the memory usage through profiling of loops. The interface between the software kernel and FleXilicon is also developed in this step. The loop mapping step maps the operations of each extracted loop body onto PEMs. The step involves various tasks such as optimization of loop bodies, assignment of operations into reconfigured instructions, scheduling of operations, and generation of a controller function (to define the functionality of the datapaths).

For the future research on the compiler for the FleXilicon architecture, several existing methods can be applied. For example, conventional compiler techniques with some modifications may be used to automate loop extraction and loop mapping procedures. Conventional HW/SW partition algorithms can be applied to select loops to be mapped onto

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

FleXilicon. Graph transformation techniques such as loop unrolling, graph merging, and tree height reduction techniques can be used to optimize the loops. Various existing high level synthesis algorithms such as scheduling, resource allocation can be employed for an optimal loop mapping.

Once loops are mapped, a controller model can be developed as a finite-state machine (FSM) (using a hardware description language) according to the loop mapping. The controller model is synthesized, converted into field-programmable gate-array (FPGA) mapping data and loaded into LUTs of the reconfigurable controller of FleXilicon.

For performance comparison, a Viterbi decoder, a 16×16 sum of absolute difference (SAD), a discrete fourier transform (DFT) block for a global positioning system (GPS), a GSM pulse shaping filter, an MP3 cosine filter were implemented on single PES using the SystemC model. The Viterbi decoder is based on soft-decision with the constraint length 9 and the 1/2 rate meeting IS-95 standard. Major critical functions such as the branch and path metrics and the add compare select (ACS) were implemented by mapping the 256-iteration loop onto the PES. The 16×16 SAD is a key operation for motion estimation in the H.264 [31], which repeats single pixel SAD operation 256 times with 256 pixel positions and accumulates the SAD values. The remaining three implementations involve MAC operations with different bit-widths and are frequently used in multimedia and wireless communications. The DFT for GPS is used to find peak energy, and it can be implemented with 8-bit $\times$ 8-bit MAC operations [34]. The GSM pulse shaping filter is used to attenuate sidelobes of GSM modulation signals, which can be implemented with 16-bit $\times$ 8-bit MAC operations [32]. Finally, the MP3 cosine filter is used for generation of sub-band audio signals for MPEG audio, which can be implemented with 16-bit $\times$ 16-bit MAC operations [33]. The three implementations are useful to compare the effectiveness of the SWP for a given architecture.

The performance of FleXilicon was compared with an ARM processor [35] and TI 320C64xx DSP chip [36]. ARM processors based on the RISC architecture are widely used as embedded processors in industry. TI 320C64xx DSP is high end DSP chips and adopts an 8-way VLIW architecture. An ARMCC compiler, an ARM emulator, a TI C compiler and a TI DSP simulator were used for compilation and cycle profiling. Profiling results for the three different architectures did not include steps for preload of initialization data.

B. Simulation Results

Simulation results for the Viterbi decoder and the 16×16 SAD operation are shown in Table III. The top entry of a cell represents the number of cycles or the execution time, and the bottom entry (in bold) is the ratio of FleXilicon to the particular item. For the Viterbi decoder, the numbers of clock cycles required for updating the state metric for one stage were profiled for the three different processors. FleXilicon requires only 12 cycles for the update, while about 16 200 cycles for the ARM processor and 3400 cycles for the TI DSP. The reduction ratio of the cycles for FleXilicon is 1351 over the ARM processor and 283 over the TI DSP. Assuming FleXilicon running at 1 GHz, the speedup for FleXilicon is about 2180 times over the ARM processor and about 235 times over the TI DSP. FleXilicon also reduces the number of clock cycles and the execution time for SAD calculations over the other two implementations, but the reduction ratios are smaller than that for the Viterbi decoder. The speedup of FleXilicon for the SAD calculations is 452 over the ARM processor and 4.73 over the TI DSP. Note that the results were obtained for single PES for FleXilicon, and the number of PESs can be scaled readily for further speedup, if needed.

TABLE III
PERFORMANCE OF FLEXILICON FOR VITERBI AND SAD OPERATIONS

		Viterbi decoder		16x16 SAD	
	Frequency	*Cycles*	*Execute Time(us)*	*Cycles*	*Execute Time(us)*
FleXilicon	1 GHz	12	0.012	13	0.013
ARM 1176	620 MHz	16,218	26.16	3,923	6.33
		1351.5	**2179.9**	**301.8**	**452.0**
TI C64xx	1.2GHz	3,399	2.83	74^{+}	0.062
		283.3	**235.8**	**5.7**	**4.73**

+ Using TI optimized library

The previous simulation results intrigued us to investigate the performance gain for FleXilicon. To the end, we analyzed the performance gain factors of FleXilicon over ARM 1176 processor for the case of the Viterbi decoder. We considered four performance factors for the Viterbi decoder, namely *op/loop, op/inst, cycles/inst*, and LLP. The "*op/loop*" indicates the number of operations necessary to update $2^8 (= 256)$ states for a 256-iterative loop. The "*op/inst*" indicates the number of operations executed per instruction for the loop, and "*cycles/inst*" the number of cycles per instruction. The LLP, as usual, denotes the number of loops which can be executed simultaneously.

The parameter values are shown in Table IV. FleXilicon improves the "*op/loop*" by five times over ARM processor. This improvement for FleXilicon is due to optimization of the loop body through operation sharing and data reuse. However, such an optimization is not feasible for the ARM processor, since the optimization requires additional instruction cycles combined with the scarcity of registers for data reuse. The "*op/inst*," the number of operations per instruction, is two for FleXilicon, since FleXilicon can execute multiple operations for single instruction using reconfigured datapath. However, it is only 0.33 for ARM 1176. In other words, execution of single Viterbi decoder operation requires execution of three instructions, on average. We noticed that many Viterbi operations are for data move, load/store, and branch operations as well as arithmetic operations, some such as load/store require multiple instruction cycles for the ARM processor. In contrast, all of those instructions can be implemented in one clock cycle for all instructions for FleXilicon. Finally, the largest gain comes from LLP, which is 32 for FleXilicon and one for the ARM processor. The four gain parameters explain the overall performance gain of 1351 for FleXilicon over ARM 1176.

Simulation results for three different MAC operations are shown in Table V. "Cycles/MAC" in the table indicates the average number of clock cycles per MAC operation, which is a good metric for architectural performance. The table shows

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

TABLE IV
PERFOMANCE PARAMETERS FOR A VITERBI DECODER

	FleXilicon	*ARM 1176*	*Improvement*
op/loop	768	3840	**5**
op/inst	2.00	0.33	**6**
cycles/inst	1.00	1.39	**1.39**
LLP	32	1	**32**
Total Cycles	12	16218	**1351.5**

TABLE V
PERFORMANCE OF FLEXILICON FOR FILTER OPERATIONS

	GPS (8b×8b)		**GSM (16b×8b)**		**MP3 (16b×16b)**	
	Cycles	*Cycles/ MAC*	*Cycles*	*Cycles/ MAC*	*Cycles*	*Cycles/ MAC*
FleXilicon	2,565	0.06	3,890	0.15	4,619	0.33
ARM 1176	209105 **81.5**	5.1	129900 **33.4**	5.15	43869+ **9.5**	3.17
TI C64xx	28,925 **11.3**	0.71	18,589 **4.8**	0.74	12,497 **2.7**	0.9

+ Using optimized assembly codes

TABLE VI
IMPLEMENTATIONS OF A CONTROLLERS

	FPGA Resource				
	# logic gates	*# Slices*	*# F/F*	*# LUT*	*Delay*
Viterbi decoder	115.9	9	10	16	1.546 ns+
16x16 SAD	76.8	8	9	14	1.144 ns+

+ xc5vlx30 speed grade -3

that FleXilicon requires less number of clock cycles over the ARM processor and the TI DSP for the three filter implementations. The speedup is larger for the GPS, which is as large as 81 over the ARM processor. This is because the GPS requires low precision multiplications (8-bit $\times$ 8-bit) to result in high degree of parallelism for FleXilicon owing to the proposed DASM scheme and the SWP. The average number of clock cycles per MAC operation is also less for FleXilicon over the ARM processor and the TI DSP, and the reduction ratio is greater for the GPS than the other two processors. The reduction ratio of the clock cycles/MAC operation is dependent on the architecture alone, not the speed of the underlying circuit. Therefore, we can state that the architecture of FleXilicon is superior to the other two competing processors, especially for low precision multiplications.

Table VI shows implementations of controllers in random logic and on an FPGA. Controllers were designed in SystemC and converted into synthesizable HDL using Synopsys. Then, it was synthesized and mapped onto a Xilinx Virtex2 FPGA using ISE tool [37]. As shown in the table, a controller for the Viterbi decoder and a 16×16 SAD block can be implemented with 116 and 77 NAND2 equivalent gates, respectively. When the two controllers are mapped onto the FPGA, each one occupies less than 3% of the available resources for a Xilinx Virtex2 xc2v40 [37], which is the smallest version of the Virtex2 family.

TABLE VII
DATA TRANSFER RATE BETWEEN MEMORIES

Apps	*External transfer rate (MB/s)*		*Internal transfer rate (GB/s)*		*Reuse Rate*	
	Ext. Read	*Ext. Write*	*Int. Read*	*Int. Write*	*Read*	*Write*
Viterbi	153.8	9.6	24.61	4.92	99.38%	99.80%
SAD	38.5	0.3	48.92	9.61	99.92%	99.997%
MP3	249.4	498.8	2.24	0.498	88.87%	0.00%
GPS	499	199.6	16.96	0.199	97.06%	0.00%
GSM	28.3	108	19.59	1.67	99.86%	93.53%

The timing analysis results for both FSMs on a Xilinx Virtex2 xc5vlx30 speed grade -3 suggest that the FSM for SAD can run up to 871 MHz and one for Viterbi 646 MHz. This implies that the FPGA speed is the limiting performance factor for FleXilicon when a conventional FPGA is employed as the controller. However, we expect that FPGAs will achieve higher clock frequency in future through process migration and optimization of LUT blocks.

C. Analysis of Data Transfer and Memory Usage

As mentioned in the earlier section, memory bandwidth is a critical design parameter affecting performance. We analyzed external data transfer rates between the memory of the host processor and local memories of FleXilicon for five applications, and internal data transfer rates between local memories and PEMs within FleXilicon. As shown in Table VII, the highest external memory transfer rate of 499 MB is required for read operations of the GSM application. The required maximal internal memory transfer rate reaches up-to 48.9 GB/s for read operations of the SAD application. As described in Section III-B, FleXilicon can support a high internal memory bandwidth of up to 64 GB/s. In fact, we noticed that FleXilicon does not cause any memory bottleneck for all the applications. The required maximal memory transfer rate is relatively low in FleXilicon, as a good portion of input and intermediate data is reused within the local memory to reduce the need for external memory transfers. Note that the required transfer rate can be achieved readily by a conventional direct memory access (DMA) or a memory bus coupled with an on-chip secondary SRAM and off-chip synchronous DRAM (SDRAM). A reuse rate is the ratio of the data transfer rate of internal transfer to that for the external transfer, and it is affected by the size of the task and the size of the local memory. The reuse rate R is calculated as $R = 1$—(External Transfer Rate/Internal Transfer Rate) and is also shown in Table VII. The table shows high reuse rates for read operations for all applications except MP3. A higher reuse rate leads to less frequent external memory accesses, which reduces the amount of the external data transfer.

Major resource necessary to run different applications on FleXilicon is memory. To assess whether the 16 kB local memories are enough, we profiled maximum memory usage of RAM and ROM data required for the five example applications, and they are shown in Table VIII. The RAM data for an application includes input stream data, intermediate results, and output

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

TABLE VIII
MAXIMUM MEMORY USAGE IN APPLICATIONS

Apps	*RAM*	*ROM*
Viterbi	0.81 KB	0.00 KB
SAD	2.50 KB	0.00 KB
MP3	3.38 KB	2.00 KB
GPS	1.75 KB	5.00 KB
GSM	0.33 KB	0.47 KB

stream data. The ROM data is for filter coefficients, which are static during the execution. The MP3 application requires the largest RAM size of 3.38 kB because of a higher tap count for the filter and the GSM of the smallest size of 0.33 kB. Based on memory usage analysis, 16 kB local memory is enough for all the five tasks, even they run simultaneously.

Finally, it is important to note that the ARM processor and the TI DSP are general purpose and programmable, while FleXilicon is targeted for specific applications, wireless communications and multimedia. So the performance comparisons reported in this section would be more favorable to FleXilicon. So the comparison results should be interpreted judiciously.

V. CONCLUSION

In this paper, we presented a new coarse-grained reconfigurable architecture called FleXilicon, which improves resource utilization and achieves a high degree of LLP. The proposed architecture mitigates major shortcomings in existing architectures through wider memory bandwidth, reconfigurable controllers, and flexible word-length support. The proposed wide bandwidth memory system enables a high degree of the LLP. The proposed reconfigurable controller addresses the shortcomings such as area inefficiency and speed overhead of existing memory based controllers. The proposed flexible word-length scheme enhances sub-word parallelism. VLSI implementation of FleXilicon indicates that the proposed pipeline architecture can achieve a high speed operation up to 1 GHz using 65-nm SOI CMOS process with moderate silicon area. To estimate the performance of FleXilicon, we implemented five different types of applications commonly used in wireless communication and multimedia applications and compared its performance with an ARM processor and a TI DSP. The simulation results indicate that FleXilicon reduces the number of clock cycles and increases the speed for all five applications. The reduction and speedup ratios are as large as two orders of magnitude for some applications.

REFERENCES

[1] R. Hartenstein, "A decade of reconfigurable computing: A visionary retrospective," in *Proc, Des., Autom. Test Eur., Conf. Exhibition*, 2001, pp. 642–649.

[2] T. J. TodMan, G. A. Constantinides, S. J. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: Architectures and design methods," *IEE Proc. Comput. Digit. Techn.*, vol. 152, no. 2, pp. 193–207, Mar. 2005.

[3] T. Miyamori and K. Olukotun, "A quantative analysis of reconfigurable comprocessors for multimedia applications," in *Proc. IEEE Symp. FPGA for Custom Comput. Mach.*, Apr. 1998, pp. 2–11.

[4] A. Sharma, C. Ebeling, and S. Hauck, "PipeRoute: A pipelining aware router for reconfigurable architectures," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 3, pp. 518–532, Mar. 2006.

[5] J. M. P. Cardoso and M. Weinhardt, "Fast and guaranteed C compilation onto the PACT-XPP reconfigurable computing platform," in *Proc. 10th Ann. IEEE Symp. Field-Program. Custom Comput. Mach.*, 2002, pp. 291–292.

[6] J. M. P. Cardoso, "On combining temporal partitioning and sharing of functional units in compilation for reconfigurable architectures," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1362–1375, Oct. 2003.

[7] S. C. Goldstein, , H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: A reconfigurable architecture and compile," *Computer*, vol. 33, no. 3, pp. 70–77, 2000.

[8] S. Cadambi and S. C. Goldstein, "Efficient place and route for pipeline reconfigurable architectures," in *Proc. Int. Conf. Comput. Des.*, Sep. 2000, pp. 423–429.

[9] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The garp architecture and C compiler," *Computer*, vol. 33, no. 4, pp. 62–69, Apr. 2000.

[10] Z. Fang, P. Tang, P. C. Yew, and C. Q. Zhu, "Dynamic processor self-scheduling for general parallel nested loops," *IEEE Trans. Comput.*, vol. C-39, no. 7, pp. 919–929, Jul. 1990.

[11] C. D. Polychronopoulous, , D.J. Kuck, and D. A. Padua, "Utilizing multidimensional loop parallelism on large-scale parallel processor systems," *IEEE Computers*, vol. C-38, no. 9, pp. 1285–1296, Sep. 1989.

[12] D. J. Lilja, "Exploiting the parallelism available in loops," *Computer*, vol. 27, no. 2, pp. 952–966, Feb. 1994.

[13] A. Chavan, P. A. Nava, and J. A. Moya, "Reconfigurable data path processor: Implementation and application for signal processing algorithms," in *Proc. IEEE 11th Digit. Signal Process. Workshop IEEE Signal Process. Education Workshop*, Aug. 2004, pp. 182–186.

[14] E. Mirsky and A. DeHon, "MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *Proc. IEEE Symp. FPGA for Custom Comput. Mach.*, 1996, pp. 157–166.

[15] T. Miyamori and K. Olukotun, "A quantitive analysis of reconfigurable coprocessors for multimedia applications," presented at the ACM/SIGDA FPGA, Monterey, CA, Feb. 1998.

[16] H. Singh, M.-H. Lee, G. Lu, F.J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Computers*, vol. 49, no. 5, pp. 465–481, May 2000.

[17] J. Becker and M. Vorbach, "Architecture, memory and interface technology integration of an industrial/academic configurable system-on-chip (CSoC)," in *Proc. IEEE Comput. Soc. Annual Symp. VLSI*, Feb. 2003, pp. 107–112.

[18] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: A computational fabric for software circuits and general purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, Mar.–Apr. 2002.

[19] D. C. Chen and J. M. Rabaey, "A reconfigurable multiprocessor IC for rapid prototyping of algorithmic-specific high-speed DSP data paths," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1895–1904, Dec. 1992.

[20] B. Salefski and L. Caglar, "Re-configurable computing in wireless," in *Proc. Des. Autom. Conf.*, 2001, pp. 178–183.

[21] J. Leijten, J. Huisken, E. Waterlander, and A. V. Wel, "AVISPA: A massively parallel reconfigurable accelerator," in *Proc. Int. Symp. Syst.-on-Chip*, 2003, pp. 165–168.

[22] K. Bondalapati, "Parallelizing DSP nested loops on reconfigurable architectures using data context switching," in *Proc. Des. Autom. Conf.*, Jun. 2001, pp. 273–276.

[23] J.-S. Lee and D. S. Ha, "FleXilicon: A reconfigurable architecture for multimedia and wireless communications," in *Proc. Int. Symp. Circuits Syst.*, May 2006, pp. 4375–4378.

[24] J. Fridman, "Sub-word parallelism in digital signal processing," *IEEE Signal Process. Mag.*, vol. 17, no. 2, pp. 27–35, Mar. 2000.

[25] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Computers*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[26] N. H. E. Weste and K. Eshraghan, *Principles of CMOS VLSI Design*. Reading, MA: Addison Wesley, 1993.

[27] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI ling addres," *IEEE Trans. Computers*, vol. 54, no. 2, pp. 225–231, Feb. 2005.

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.

[28] Synopsys, Inc., Mountain View, CA, "DesignWare library user's guide," [Online]. Available: http://www.synopsys.com/
[29] Synopsys, Inc., Mountain View, CA, "PrimeTime user's guide," [Online]. Available: http://www.synopsys.com/
[30] "SystemC user's guide," [Online]. Available: http://www.system.org/
[31] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[32] Modulation, "GSM Recommendation 05.04," Dec. 1999.
[33] MPEG-1, "ISO CD 11172-3: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mb/s," Nov. 1991.
[34] U. Cheng, W. J. Hurd, and J. I. Statman, "Spread-spectrum code acquisition in the presence of Doppler shift and data modulation," *IEEE Trans. Commun.*, vol. 38, no. 2, pp. 241–250, Feb. 1990.
[35] ARM Ltd., Cambridge, U.K., "ARM1176 technical reference manual," [Online]. Available: http://www.arm.com/
[36] Texas Instruments, Dallas, TX, "TMS320C6455 datasheet," [Online]. Available: http://www.ti.com/
[37] Xilinx, Inc., San Jose, CA, "Xilinx product datasheets," [Online]. Available: http://www.xilinx.com/
[38] M. J. Myjak and J. G. Delgado-Frias, "Pipelined multipliers for reconfigurable hardware," in *Proc. Parallel Distrib. Process. Symp.*, Apr. 2004, pp. 26–30.
[39] S. D. Hyanes, , A. B. Ferrari, and P. Y. Cheung, "Algorithms and structures for reconfigurable multiplication units," in *Proc. Integr. Circuit Des.*, Oct. 1998, pp. 13–18.
[40] K. Biswas, H. Wu, and M. Ahmadi, "Fixed-width multi-level recursive multipliers," in *Asilomar Conf. Signals, Syst. Comput.*, Oct. 2006, pp. 935–938.

Jong-Suk Lee was born in Daegu, Korea, in 1973. He received the B.S. and M.S. degrees in electrical engineering from Sogang University, Seoul, Korea, in 1995 and 1997, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering from Virginia Polytechnic Institute of and State University, Blacksburg.

He is currently working as an implementation engineer for AMD Inc., Boxborough, MA. He was working for Samsung Electronics Company, where he was engaged in the development of 1-GHz ALPHA CPU with Digital Equipment Corp. (DEC). He was a Verification and Integration Engineer for Qualcomm Inc., San Diego, CA. His research interests include high-speed VLSI design, microprocessor architecture, and SOC design.

Dong Sam Ha (M'86–SM'97–F'08) received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 1974 and the M.S. and Ph.D. degrees in electrical and computer engineering from University of Iowa, Iowa City, in 1984 and 1986, respectively.

Since Fall 1986, he has been a faculty member with the Department of Electrical and Computer Engineering, Virginia Polytechnic Institute of and State University, Blacksburg. Currently, he is a Professor and Director of the Center for Embedded Systems and Critical Applications (CESCA). He supervises the Virginia Tech VLSI for Telecommunications (VTVT) Group, which specializes in low-power VLSI design for various applications including wireless communications. Along with his students, he has developed four computer-aided design tools for digital circuit testing and has created CMS standard cell libraries. The library cells and the source code for the four tools have been distributed to over 280 universities and research institutions worldwide. His research interests include structural health monitoring system, low-power VLSI design for wireless communications, low power General Chair analog and mixed-signal design, RF IC design, UWB RFIDs, and reconfigurable architectures.

Dr. Ha was General Chair of the System-on-Chip Conference (SOCC) in 2005 and Technical Program Chair of the same conference in 2003 and 2004.

Authorized licensed use limited to: to IEEExplore provided by Virginia Tech Libraries. Downloaded on August 4, 2009 at 13:05 from IEEE Xplore. Restrictions apply.