A Single-Chip Programmable Platform Based on a Multithreaded Processor and Configurable Logic Clusters

Young-Don Bae, *Student Member, IEEE*, Seong-Il Park, *Student Member, IEEE*, and In-Cheol Park, *Senior Member, IEEE*

***Abstract*—This paper presents a single-chip programmable platform that integrates most of hardware blocks required in the design of embedded system chips. The platform includes a 32-bit multithreaded RISC processor (MT-RISC), configurable logic clusters (CLCs), programmable first-in-first-out (FIFO) memories, control circuitry, and on-chip memories. For rapid thread switch, a multithreaded processor equipped with a hardware thread scheduling unit is adopted, and configurable logics are grouped into clusters for IP-based design. By integrating both the multithreaded processor and the configurable logic on a single chip, high-level language-based designs can be easily accommodated by performing the complex and concurrent functions of a target chip on the multithreaded processor and implementing the external interface functions into the configurable logic clusters. A 64-mm^2 prototype chip integrating a four-threaded MT-RISC, three CLCs, programmable FIFOs, and 8-kB on-chip memories is fabricated in a 0.35-μm CMOS technology with four metal layers, which operates at 100-MHz clock frequency and consumes 370 mW at 3.3-V power supply.**

***Index Terms*—Hardware platform, multithreading, platform-based design, programmable logic, system-on-chip design.**

I. Introduction

As DESIGN complexity increases significantly according to the trend of system-on-a-chip (SoC), design time and cost become dominant factors in the decision making process. Much research has focused on reducing the design time and cost by design reuses. Although the design time is significantly reduced by reusing predeveloped intellectual properties (IPs), integrating and verifying IPs still takes lots of design time. In order to reduce the integration and verification time, it is necessary to define a communication protocol. This implies that a basic microarchitecture is defined for a range of applications, and major components such as microprocessors and buses are always contained in every implementation, whereas other blocks are customized according to a given application. Such a microarchitecture is called a *platform* [1], [2].

Manuscript received November 21, 2002; revised June 3, 2003. This work was supported by the Korea Science and Engineering Foundation through the MICROS Center, by the Ministry of Science and Technology and the Ministry of Commerce, Industry, and Energy through the Project System IC 2010, and by the IC Design Education Center (IDEC).

The authors are with the Division of Electrical Engineering, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Korea (e-mail: ydbae@ieee.org; icpark@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/JSSC.2003.817259

The evolution of platform-based design has given rise to various platform types ranging from full-application platforms to fully programmable platforms. As fully programmable platforms containing a microprocessor and programmable logic blocks make it possible to customize both the hardware and software, many field-programmable gate array (FPGA) providers are now adding embedded processors to their programmable logic products [3]–[6].

However, these platforms fail to notice some important aspects of system design. First, in order to implement complex embedded systems, the problems associated with concurrent processing and communication must be dealt with in a uniform and scalable manner. Second, the traditional programmable logic architectures do not coincide with the IP-based design strategy. IPs are intermingled at the time of placement and routing as the architecture is based on a flattened array of logic blocks, leading to many iterations of placement and routing in timing verification.

In this paper, we present a new configurable device called a single-chip programmable platform (SPP) to provide concurrent processing by employing a multithreaded RISC processor and to support IP-based design with configurable logic clusters.

II. Overall Architecture

Fig. 1 shows a block diagram of the proposed SPP. A 32-bit multithreaded RISC processor (MT-RISC) is integrated to execute multiple threads concurrently. A hardware unit of thread scheduling is implemented to reduce the overhead of software thread scheduling. Configurable logic blocks (CLBs) are grouped into clusters, each of which is called a configurable logic cluster (CLC) and can be programmed for an IP or a custom logic. Programmable FIFOs temporarily store the data to be transferred between the MT-RISC and CLCs. On-chip memories are also integrated for the program and working memory.

The major objective of the SPP is to accommodate high-level language (HLL)-based design. In the conventional approaches, a design modeled in an HLL should be synthesized into a gate-level netlist. Although the system functionality is already proved with the HLL simulation, the gate-level translation is very time consuming and error prone, and furthermore, the verification of the gate-level netlist is difficult. To enable HLL-based system design without gate-level translation, the proposed SPP consists

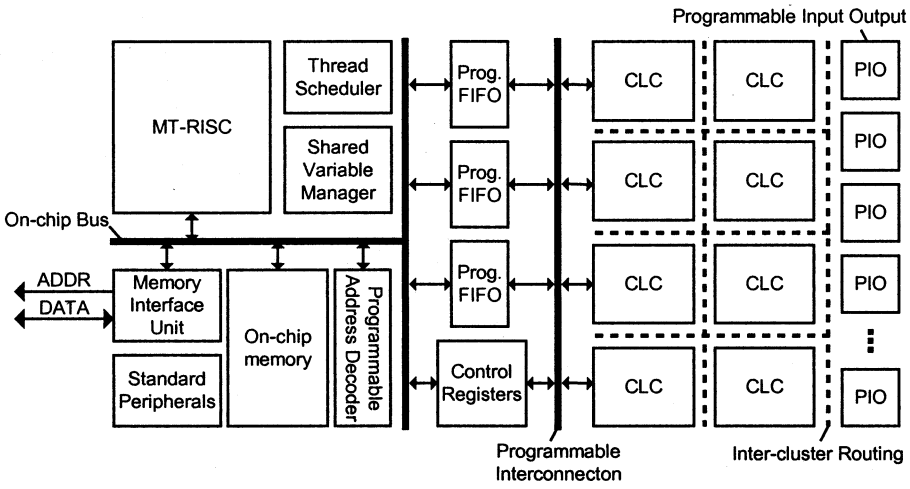

Fig. 1. Block diagram of SPP.

of a multithreaded microprocessor to perform the concurrent system behavior described in an HLL, configurable logic clusters to process external interface functions required to deal with off-chip communication protocols, and a closely coupled communication architecture between the multithreaded processor and configurable logic clusters.

The system behavior described in an HLL is compiled and executed by the microprocessor. A conventional microprocessor can perform the same functionality as a hardware unit realized with dedicated circuits, but can cause significant performance degradation as the hardware consists of concurrent operations by its nature, while the microprocessor executes instructions sequentially. In embedded systems, the system behavior is usually decomposed into multiple concurrent tasks, and it is very hard to model the concurrent and complex behavior of the target chip sequentially. Although conventional microprocessors can achieve multitasking with the assistance of a real-time operating system supporting multithreading, tens of clock cycles are wasted to save and restore the context and another tens of clock cycles to determine the next task to be switched to, according to the condition of shared resources and the status of the tasks [7]. The proposed SPP is equipped with a multithreaded RISC processor to virtually eliminate the switching overhead of the concurrent behavior, and a hardware thread scheduler to reduce the scheduling overhead [8]. The hardware thread scheduler decides the next task based on the priorities of ready tasks and provides interthread communication and synchronization as in real-time operating systems. Recently, multithreaded architectures have been employed to meet the requirement of high-performance and parallel data processing. Intel's network processors contain several 32-bit independent multithreaded microengines that have an independent program counter for each thread [9]. Although they reduce the overhead of context switching, a significant amount of computing power can be wasted as they rely on software for scheduling and communicating among threads.

Although an HLL is a powerful way to describe the functional behavior of the target chip, it is not adequate for describing the behavior of external interfaces. An external interface consists of multiple input–output (IO) signals that must be generated according to the specified timing constraints. This behavior can be programmed in a microprocessor operating at a much higher frequency than that of the external interface, as a transaction that can be performed in an external cycle must be programmed with a sequence of instructions. Even worse, the processor may be dedicated to performing the interface signal generation. Therefore, it is more effective to provide a custom logic that translates IO signals into a data stream that the microprocessor can read easily and generates IO signals from a data stream written by the microprocessor. The custom logic designed with an HDL for the given protocol is configured into the on-chip programmable logic. Since widely used external interfaces are usually standardized, we can make an IP library for common external interfaces. If an external interface is needed, an IP can be selected from the library, instead of designing it each time, and placed onto the programmable logic. The on-chip programmable logic is specially designed to incorporate IP-based design by providing cluster-level isolation. This feature enables designers to program each IP separately.

For example, let us consider the design of a single-chip video decoder, shown in Fig. 2(a). In the conventional approach, each block is realized as a dedicated hardware as shown in Fig. 2(b). Using the SPP, the functional model of each computation-intensive block is executed in a thread of the MT-RISC and the main controller also mapped to a thread. External interfaces such as NTSC, YUV, and I^2C are mapped to CLCs separately and programmable FIFOs are used to transfer data between the external interfaces and IO threads.

III. MT-RISC Architecture

As the main processor has to handle concurrent IO events and data as well as system controls, fast context switching is essential. In conventional embedded processors, the context switching is slow, often requiring several hundreds of clock cycles. Thus, the proposed SPP employs a multithreaded processor to allow rapid processing of concurrent events. The architecture is designed to support up to fifteen threads. As shown in Fig. 3, major differences from the conventional RISC architecture are found in the register file, the address generation logic including program counters, and the hardware thread

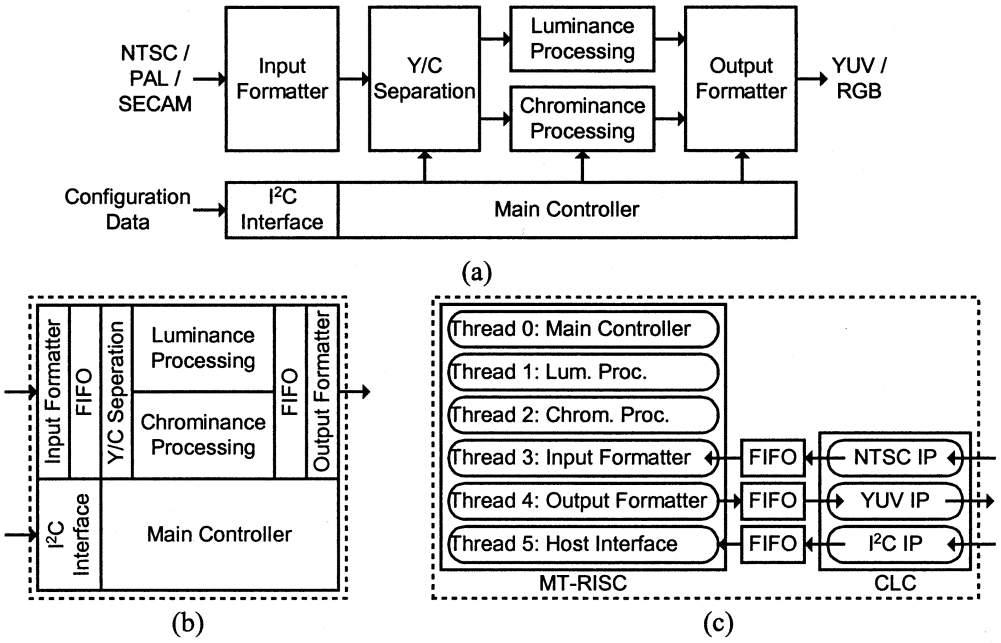

Fig. 2. System design of (a) a single-chip video decoder using (b) a conventional and (c) the proposed approach.

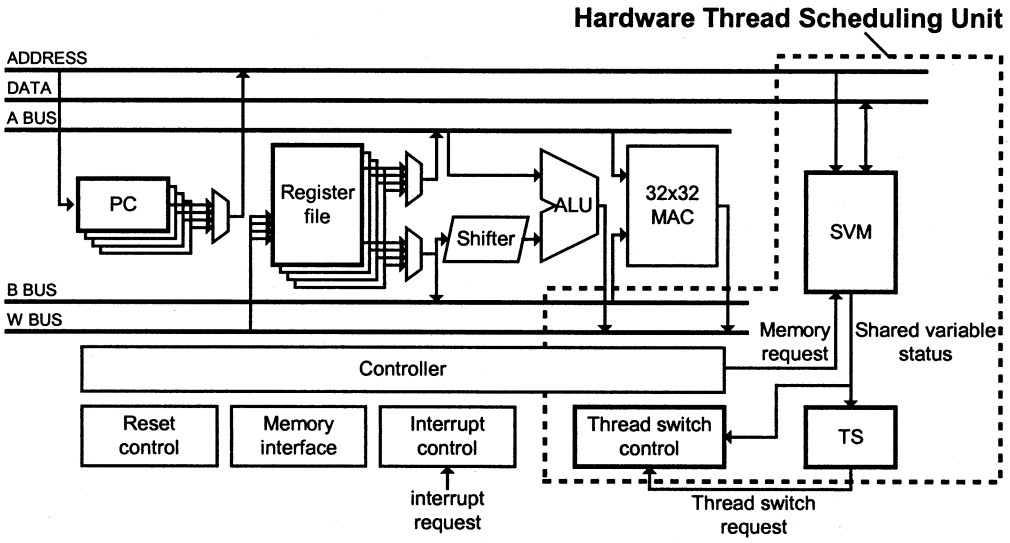

Fig. 3. Architecture of the MT-RISC.

scheduling unit. The register file contains general-purpose registers and a stack pointer, and a link register and a status register for each thread. The MT-RISC performs context switching by multiplexing these sets of registers. Compared with a single-threaded processor that has the same basic instructions, the context-switching overhead is reduced to 3% in our experiment.

The hardware thread scheduling unit consists of a shared variable manager (SVM) and a thread scheduler (TS) to prevent the loss of computing power resulting from the temporal unavailability of shared variables. The SVM monitors all the read and write operations related to shared variables, and the TS decides the next thread to be executed by taking into account the status of shared variables. A thread switch takes place when the thread being executed consumes more than a given amount of time or another thread with a higher priority becomes ready to proceed. If a shared variable becomes ready, the SVM becomes aware of which threads become ready as it manages all the information of shared variables including the identifiers of producer and consumer threads. Moreover, if a thread cannot proceed because its resource is not ready, the TS raises the priority of the thread that can make the resource ready (priority inheritance). All the memory access instructions are designed to interact with the SVM. The instructions are equivalent to conventional load and store instructions when they access nonshared variables. However, if they access a shared variable that is not ready, a thread switch takes place, as shown in Fig. 4, which depicts two cases of a load operation.

For the *load*, the processor generates a target address on the address bus. Then the SVM indicates the condition of the vari-

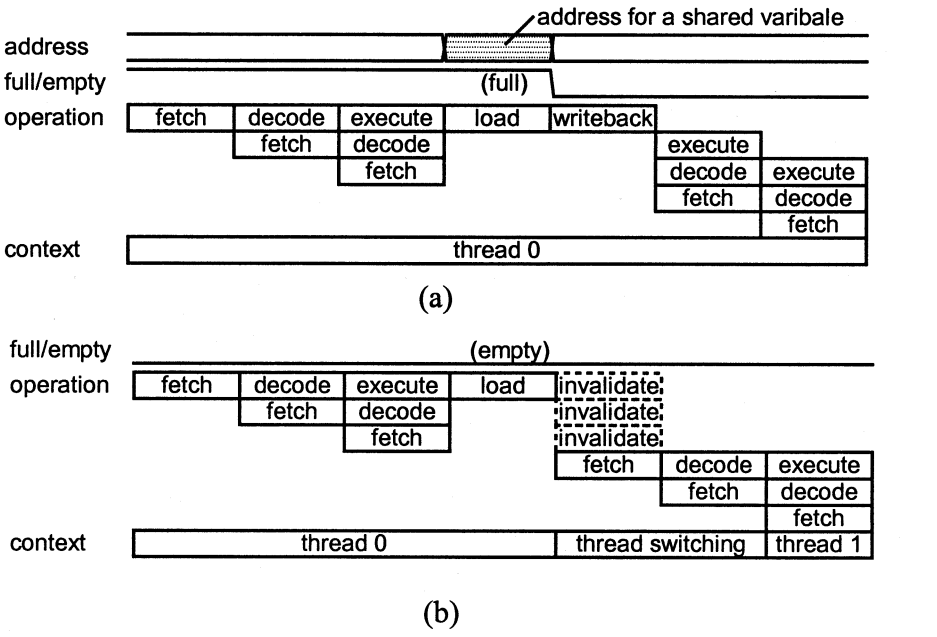

Fig. 4. Conditional thread switching during a load operation when the shared variable is (a) ready and (b) not ready.

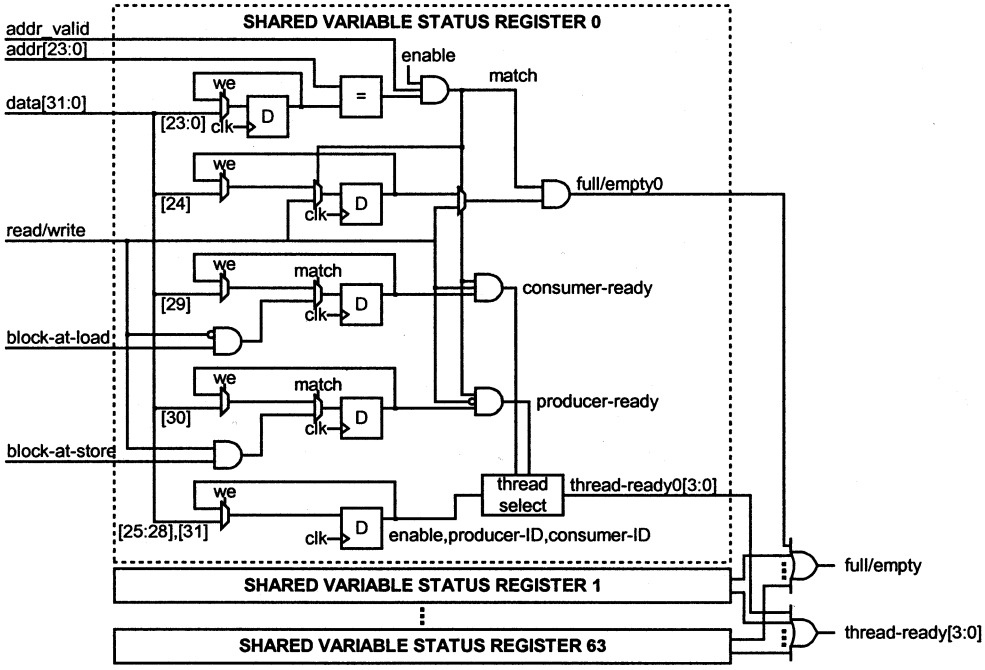

Fig. 5. Circuit diagram of the shared variable manager.

able corresponding to the address. If the address is not related to any shared variable or the corresponding shared variable is ready to be accessed, the processor pipeline proceeds normally, as shown in Fig. 4(a). However, if the shared variable is not ready, the processor immediately switches to the next thread determined by the TS. If a thread switch arises, the pipeline is invalidated for the current thread and filled with the instructions of the following thread, as shown in Fig. 4(b). In this way, the synchronization among threads is achieved implicitly, which makes the high-level description easy as the designer does not need to take care of the detailed synchronization procedure. As shown in Fig. 5, the SVM consists of 64 shared variable status registers, each of which stores the identifiers of the producer and consumer threads and the status of the shared variable such as *block-at-load* and *block-at-store*.

Experimental results show that the processor efficiency of a single-threaded processor degrades significantly as the number of IO tasks increases, as shown in Fig. 6. Assume that a thread is bounded to an IO and every IO has a constant data rate of N_{IOevent}. For n IOs, the processor efficiency $E(n)$ can be defined as given in the equation shown at the bottom of the next page, where N_{switch} is the number of thread switches for

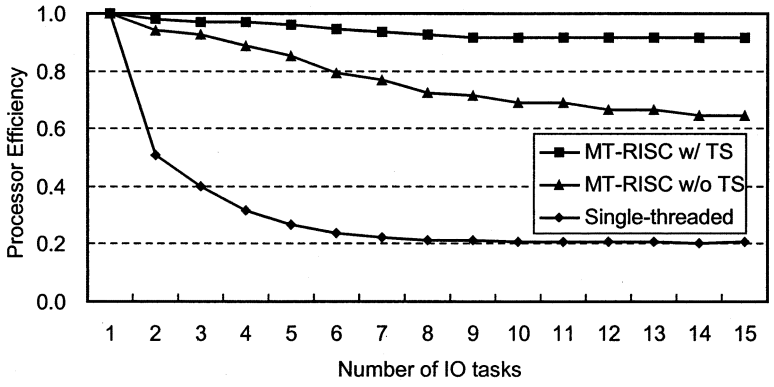

Fig. 6. Processor efficiency versus the number of IO tasks.

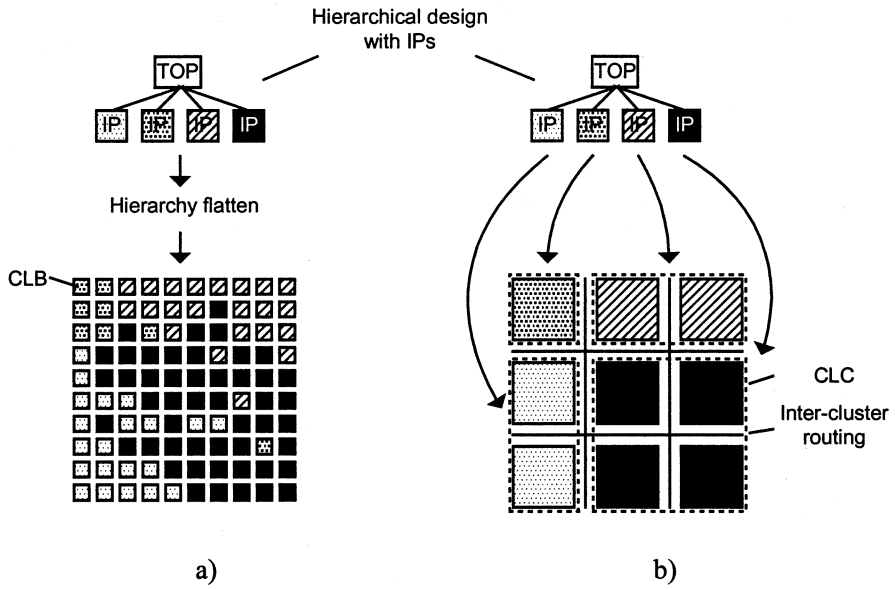

Fig. 7. IP-based design based on (a) an array-based FPGA architecture and (b) the configurable logic cluster architecture.

n IO tasks and $C_{\mathrm{IOevent}}, C_{\mathrm{synch}}, C_{\mathrm{switch}}$, and C_{schedule} are the number of cycles for processing an event, checking the status of shared resources, switching, and determining the next thread, respectively. Fig. 6 is evaluated by assuming that each IO has a data rate of 1M words per second to be moved to the register file and the microprocessor is operating with a 100-MHz processor clock. When it processes fifteen IO tasks, for example, 80% performance is wasted by the context switching and scheduling, whereas the MT-RISC maintains more than 90% efficiency.

IV. CLC Architecture

IP-based design is essential for SoC design as preverified IPs can reduce the design time significantly. When we apply the same concept to FPGA-based design, we can also expect the reduction of design time. However, the conventional FPGA design methodology does not easily coincide with the IP-based design concept in several aspects. In the conventional FPGA design methodology, a hierarchical design, often including IPs, is flattened to partition it into small-sized logic blocks [10], [11]. Then, the logic blocks are placed into CLBs and routed. As the timing characteristics of IPs are not preserved during the processes, it is difficult to guarantee the correct timing behavior of the IPs. In addition, if we change a small portion of the design, we have to repeat the whole process. Even if we do not modify any IP, we have to place and route the whole design including the IPs. Moreover, we have to verify not only the modified portion, but also the rest, as the changes caused by the placement and routing affect the timing characteristics of IPs,

$$E(n) = \frac{n \times N_{\mathrm{IOevent}} \times C_{\mathrm{IOevent}}}{n \times N_{\mathrm{IOevent}} \times (C_{\mathrm{IOevent}} + C_{\mathrm{synch}}) + N_{\mathrm{switch}}(n) \times (C_{\mathrm{switch}} + C_{\mathrm{schedule}})}$$

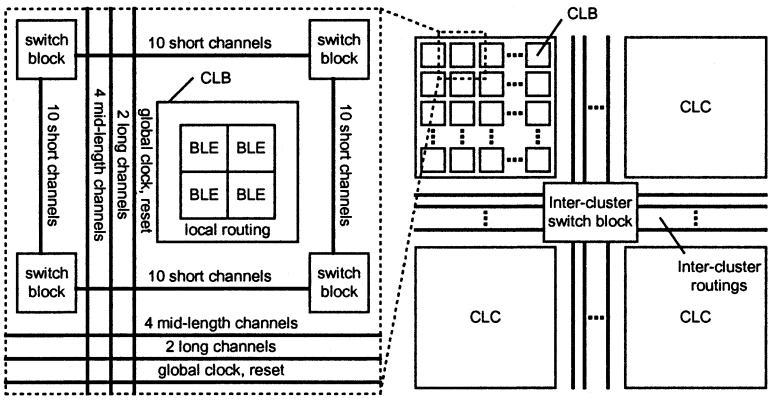

Fig. 8. Structure of configurable logic clusters.

increasing design time considerably. Fig. 7 shows a conventional FPGA architecture and the proposed CLC architecture.

If we can preserve the placement and routing information of IPs, we can guarantee their timing behavior and thereby reduce design time. For this, a group of CLBs mapped to an IP should be isolated from the others. Compared with the conventional FPGA architecture that does not support the isolation of CLBs, the CLC architecture mitigates the drawbacks by providing cluster-level isolation.

The structure of a CLC consists of 64 CLBs that can be connected with ten short channels, four mid-length channels, and two long channels, as shown in Fig. 8. Each CLB contains four basic logic elements (BLEs) comprising a combinational logic and a flip-flop. The combinational logic is essentially made of an SRAM lookup table (LUT) as the method offers the maximum flexibility in implementing combinational functions. A LUT has four inputs and one output whose value is available directly or can be fed to the flip-flop, and thus any logic function of up to four variables can be implemented. The flip-flop is a D-type flip-flop with a synchronous reset. Four BLEs forming a CLB have twelve inputs and four outputs that are accessible by all the BLEs inside the CLB via the local routing channels and are connected to vertical and horizontal routing channels through the connection blocks. The detailed circuit diagram of a CLB is shown in Fig. 9. Each CLC can be configured separately and adjacent CLCs can be connected via intercluster switch blocks to accommodate a large and complex IP.

To evaluate the CLC architecture, we designed several experimental systems by combining a number of circuits quoted from the MCNC (Microelectronics Center of North Carolina) benchmark suit [12]. Compared with the conventional array-based architecture, the CLC architecture reduces design time by 86% with area overhead of 22% on the average, as shown in Table I, where the area overhead denotes the number of configurable logic blocks that are not used and not occupied by other IPs, and *dalu, sbc, c1908, mm9a,* and *mult32a* are a dedicated ALU, a snoop bus controller, an error-correcting logic, a minmax circuit, and a 32-bit multiplier, respectively. Most of the overhead is caused by small-sized designs that are mapped to only one cluster. For medium- and large-sized designs, the overhead is not significant. Although this approach may result in area overhead, contemporary high-density FPGAs of several million gates make easy integration and design time reduction more important than area optimization.

TABLE I
EXPERIMENTAL RESULTS FOR MCNC BENCHMARKS

Description	Area Overhead	Design Time Reduction
dalu + sbc + c1908 + mm9a	16 %	87 %
dalu + sbc + c1908 + mult32a	18 %	93 %
dalu + sbc + mm9a + mult32a	25 %	86 %
dalu + c1908 + mm9a + mult32a	24 %	84 %
sbc + c1908 + mm9a + mult32a	27 %	81 %
Average	22 %	86 %

V. PROGRAMMABLE FIFO

As the IO signals change frequently, the MT-RISC should access them fast enough not to miss the information. Although employing a multithreaded processor can reduce the processor's response time to the IO event, it is almost impossible to guarantee the response time as a thread of higher priority can preempt the IO thread. Programmable FIFOs are integrated to hold the IO information temporarily when the MT-RISC cannot access it immediately. Without the FIFO, the data rate on an IO is limited by the worst case response time of the MT-RISC. Given an N-entry FIFO, the data rate can be equal to the average response time of N transactions.

A programmable FIFO can be configured to operate as a FIFO or a RAM. In the FIFO mode, it temporarily stores and retrieves the data to be transferred between the MT-RISC and the CLCs in a first-in-first-out manner. In the RAM mode, it is used for a scratchpad memory. Data are stored in or retrieved from the memory by providing the corresponding address directly. The mode of a FIFO can be changed by configuring the control registers in the FIFO.

As shown in Fig. 10, the multiplexer selects an operation mode by coupling either the address bus or the output of read/write address counters to the read/write address busses of the RAM. In the RAM mode, the address bus is directly used to determine the memory location of data to be stored and retrieved. In the FIFO mode, the address is provided by the read address counter or the write address counter. Two or more FIFOs can be combined to make a larger FIFO or RAM. For example, four programmable FIFO blocks, each of which has a

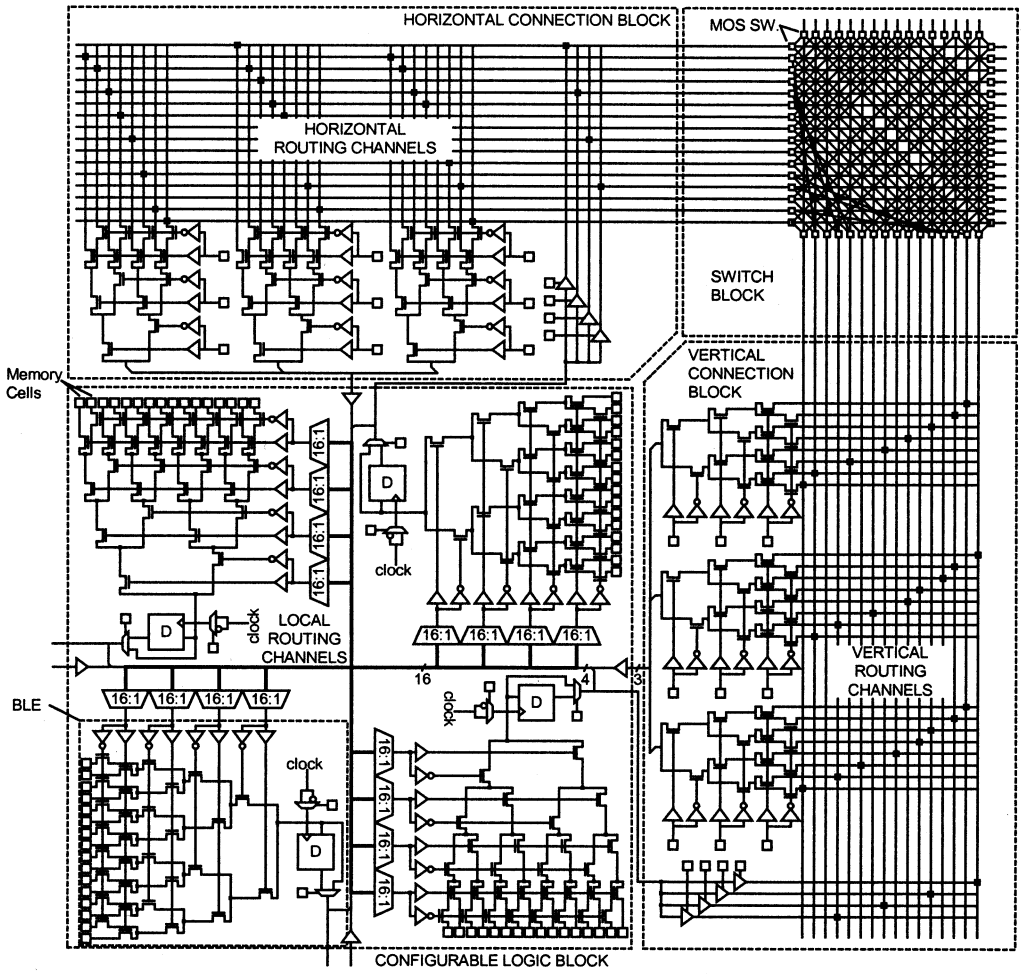

Fig. 9. Circuit diagram of a CLB.

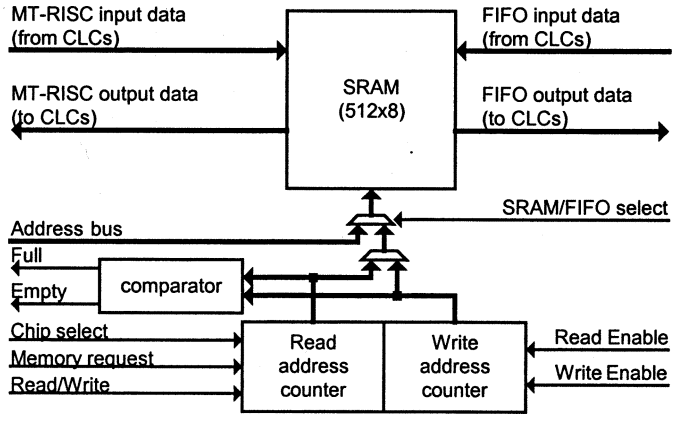

Fig. 10. Programmable FIFO.

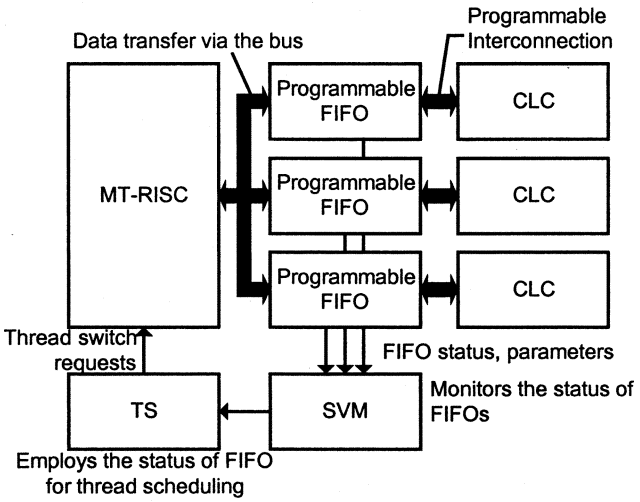

Fig. 11. Dynamic control of data transfer.

512×8 memory in the prototype chip, can be configured as a 512×32, a 2048×8 or two 512×16 memories.

The programmable FIFO informs the MT-RISC of its status to prevent invalid accesses such as reading from an empty FIFO and writing to a full FIFO. To determine the status, a FIFO compares the values of the read address and write address counters. If the difference is less/greater than a given value, the FIFO is nearly full/empty. The values are programmable in the application software to make it know how full/empty the FIFO is when it is warned of the nearly full/empty condition by the hardware. All of the FIFO accesses are also monitored by the SVM so that the TS takes into account the status of FIFOs, as shown in Fig. 11. If a FIFO becomes nearly full, the TS increases the priority of the consumer thread. Likewise, when the FIFO is nearly empty, the TS increases the priority of the producer thread. In

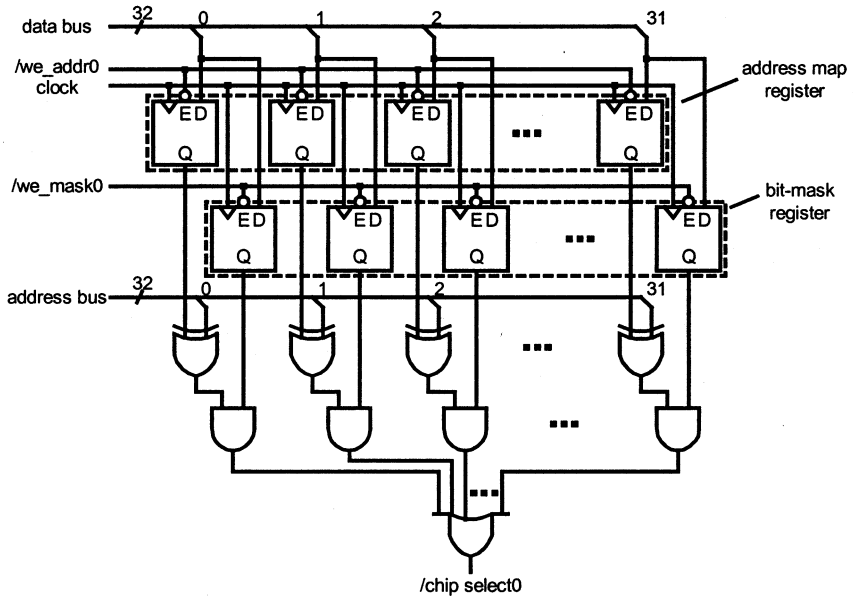

Fig. 12. Programmable address decoder.

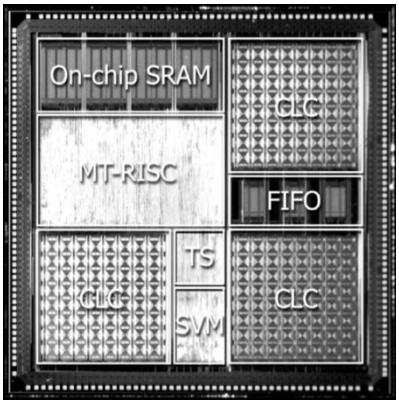

Fig. 13. Micrograph of the prototype chip.

this way, the data transfer among the MT-RISC and the CLCs is dynamically controlled depending on the FIFO status. This feature is very useful to meet a given latency or throughput.

VI. Programmable Address Decoder

In order to achieve flexibility in hardware configurations, a programmable address decoder is integrated as shown in Fig. 1. An IP mapped to a CLC is regarded as a memory-mapped device that the MT-RISC can access as a memory. The programmable address decoder allocates a memory region for an IP. As shown in Fig. 12, two control registers are used to specify a memory region. The address map register specifies the memory location of an IP and the bit-mask register specifies the address bits to be compared. If an address provided by the MT-RISC corresponds to the memory region specified by the control registers, the chip select signal is activated to allow the MT-RISC to access the IP.

VII. Implementation Results

A prototype chip including a four-threaded MT-RISC, three CLCs, three FIFO blocks, and an 8-kB on-chip SRAM was fabricated in a 0.35-μm CMOS technology with four metal layers. The on-chip SRAM is integrated for program and working memories. The CLCs were designed in a full-custom manner and the entire design was modeled and simulated in Verilog HDL. The RAM cells were built by using a memory generator. A micrograph of the chip is shown in Fig. 13 and its characteristics are summarized in Table II. The chip occupies 8×8 mm^2 including pads, and a single power supply of 3.3 V

TABLE II
Prototype Chip Characteristics.

Technology	0.35-μm CMOS with 4 metal layers
Chip size	8 x 8 mm^2 (1.2 million TRs)
Power Supply	3.3 V
Clock frequency	100MHz for MT-RISC 50MHz for CLC
Power consumption	370mW
Features	Four-threaded RISC Three CLCs (about 10,000 gates) Three FIFO (512 entries each) 8-KB on-chip SRAM

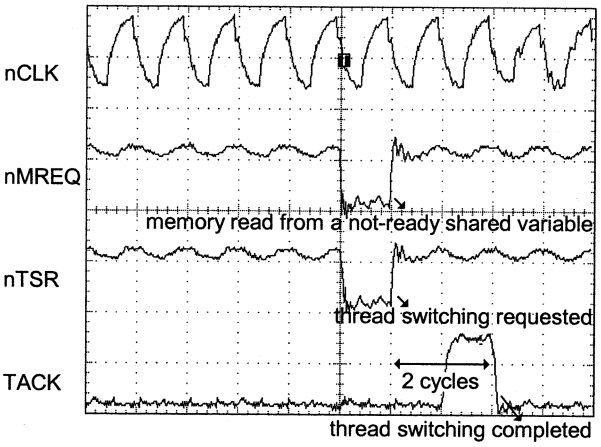

Fig. 14. Measured waveforms.

is used for logic core and IO pads. The MT-RISC operates at 100-MHz clock frequency, resulting in 110 Dhrystone 2.1 MIPS. Three CLCs can deal with about 10 000 logic gates. Experimental results for MCNC benchmarks show the CLC operates at 50 MHz. Fig. 14 shows the waveforms measured during a thread switch. Signals nTSR (thread switch request) and nMREQ (memory request) were asserted because the MT-RISC attempted to read a shared variable that was not ready. After two cycles, the MT-RISC started to execute another thread and signal TACK (thread switch acknowledge) was activated to indicate that the thread switch was completed.

VIII. Conclusion

In this paper, we have presented a single-chip programmable platform to cope with the design time and cost problems of embedded system design. The SPP employs a 32-bit multithreaded processor with a hardware thread scheduling unit, configurable logic clusters for IP-based design, and a closely coupled communication structure based on programmable FIFOs. The multithreaded processor reduces the context-switching time by 97% compared to a conventional single-threaded processor and achieves up to 70% performance enhancement in processing concurrent tasks. The configurable logic cluster is proposed to preserve the timing characteristics of IPs during the placement and routing by providing cluster-level isolation, and thereby reduces the design time by 86% in our experiments on the average. The programmable FIFOs enable fast data transfer between the processor and IPs mapped to the configurable logic clusters. In addition, a programmable address decoder is also embedded to provide flexible hardware configurations. A prototype chip including a four-threaded MT-RISC, three CLCs, three programmable FIFOs, and 8-kB on-chip memories was fabricated in a 0.35-μm CMOS technology with four metal layers to verify the effectiveness of the proposed platform.

References

[1] Y.-D. Bae, S.-I. Park, Y.-S. Lee, and I.-C. Park, "A single-chip programmable platform based on a multithreaded processor and configurable logic clusters," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2002, pp. 336–337.

[2] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1523–1543, Dec. 2000.

[3] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh, "A quick safari through the reconfiguration jungle," in *Proc. ACM/IEEE DAC*, June 2001, pp. 172–177.

[4] S. Winegarden, "Bus architecture of a system on a chip with user-configurable system logic," *IEEE J. Solid-State Circuits*, vol. 35, pp. 425–433, Mar. 2000.

[5] (2002) Excalibur Device Overview Data Sheet. Altera Corp. [Online]. Available: http://www.altera.com/literature/ds/ds_arm.pdf

[6] (2003) Virtex-II Pro Platform FPGAs: Complete Data Sheet. Xilinx Corp. [Online]. Available: http://direct.xilinx.com/bvdocs/publications/ds083.pdf

[7] P. R. Nuth and W. J. Dally, "A mechanism for efficient context switching," in *Proc. ICCD*, Sept. 1991, pp. 301–304.

[8] S. Storino, A. Aipperspach, J. Borkenhagen, R. Eickemeyer, S. Kunkel, S. Levenstein, and G. Uhlmann, "A commercial multithreaded RISC processor," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 1998, pp. 234–235.

[9] (2002) Intel IXP2850 Network Processor Product Brief. Intel Corp. [Online]. Available: ftp://download.intel.com/design/network/ProdBrf/25213601.pdf

[10] S. Kaptanoglu, G. Bakker, A. Kundu, I. Corneillet, and B. Ting, "A new high density and very low cost reprogrammable FPGA architecture," in *Proc. ACM/SIGDA FPGA*, 1999, pp. 3–12.

[11] Y.-T. Lai and P.-T. Wang, "Hierarchical interconnection structures for field programmable gate arrays," *IEEE Trans. VLSI Syst.*, vol. 5, pp. 186–196, June 1997.

[12] K. Kozminski, "Benchmarks for layout synthesis—Evolution and current status," *Proc. ACM/IEEE DAC*, pp. 265–270, June 1991.

Young-Don Bae (S'01) received the B.S. and M.S. degrees in electronics engineering from Chungnam National University, Daejeon, Korea, in 1997 and 1999, respectively. Currently, he is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejon.

His research interests include system-on-a-chip design methodology and high-performance and low-power microprocessor design.

Seong-Il Park (S'98) received the B.S. degree in electronics engineering from Korea University, Seoul, Korea, in 1996, and the M.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1998. Currently, he is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science, KAIST.

His research interests include VLSI design for communication and multimedia applications.

In-Cheol Park (S'88–M'92–SM'02) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1988 and 1992, respectively.

Since June 1996, he has been with the Department of Electrical Engineering and Computer Science, KAIST, first as an Assistant Professor and now an Associate Professor. Prior to joining KAIST, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, from May 1995 to May 1996, where he researched high-speed circuit design. His current research interests include computer-aided design algorithms for high-level synthesis and very large scale integration architectures for general-purpose microprocessors.

Dr. Park received the Best Paper Award from the ICCD in 1999 and the Best Design Award from the ASP-DAC in 1997.