Title	Multi-tasking of fuzzy inference processor through real-time context switching
Author(s)	Cao, Qi; Song, Liqin; Shi, Xiaomeng; Li, Ju Hui
Citation	Cao, Q., Song, L., Shi, X., & Li, J. H. (2009). Multi-tasking of fuzzy inference processor through real-time context switching. IEEE/ASME International Conference on Advanced Intelligent Mechatronics (:2009:Singapore)
Date	2009
URL	http://hdl.handle.net/10220/6290
Rights	

2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics
Suntec Convention and Exhibition Center
Singapore, July 14-17, 2009

Multi-Tasking of Fuzzy Inference Processor Through Real-Time Context Switching

Qi Cao, Liqin Song, Xiaomeng Shi and Ju Hui Li

Abstract— **We present a multi-task fuzzy inference processor (MtFIP) in this paper. A reconfigurable fuzzy inference chip (RcFIP) with 17 mega fuzzy logic inference per second (FLIPS) capable of supporting real time context switching has been proposed in our earlier work. High performance similar to that of the RcFIP is attainable by the MtFIP. Each task is executed by a RcFIP configured specific to the domain of the application. High flexibility is achieved by integrating several RcFIPs in the MtFIP. The compact architecture of the MtFIP allows for seamless online context switching among multiple tasks. The hardware architecture of the MtFIP is described in detail in this paper.**

I. Introduction

With good flexibility and performance, it is an interesting research topic for processors with multi-tasking capability. In general, there are two design approaches for multi-task chips; multi-processor in a single chip, and single processor with multi-tasking capability. For the former approach, tasks are handled by multiple processors. Each dedicated processor in the chip serves a specific task. Current system-on-chip (SoC) technology shows a trend toward high level of integration, making multi-processor cores in a single chip feasible [1]. Paulin *et al.* [2] presented a domain-specific multi-processor SoC architecture. Theelen *et al.* [3] proposed a scalable multi-microprocessor architecture consisting of a number of identical master processors and configurable co-processors. Rutten *et al.* [4] designed a heterogeneous multi-processor architecture template for digital multimedia applications. The multi-processor SoC architectures with high performance are able to meet the needs of application-specific requirements. The disadvantage is that the circuit becomes quite complex due to the integration of several sub-systems into a complete multi-processor system.

For the latter approach, multiple tasks are manipulated by a single processor with multi-tasking capability. Such single multi-task processor could be a general purpose processor. Great flexibility is offered by general purpose processors for different tasks. However, the processing speed of general purpose processor architecture may not be suitable for some applications with strict performance requirements [5], [6]. The development of special purpose or application-specific processor architecture is more appropriate to cater for the design constraints [7], [8]. In this respect, fuzzy inference processor is a good candidate, since it is able to offer high execution speed for time-critical applications [9], [10].

For multi-task systems, context switching within the same application is usually faster with less communication overhead. However, dynamic context switching among different applications is often expensive. In our work, based on the design idea of the RcFIP [11], a real-time MtFIP capable of online context switching among different applications is proposed. Multiple tasks share most of the hardware resources within a compact structure. To the best of our knowledge, there are no such fuzzy inference processors reported in literature.

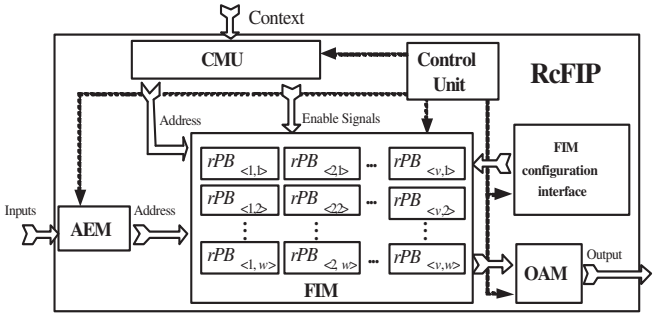
The remaining of this paper is organized as follows. In the next section, we give an overview of previous works on fuzzy inference processors. In Section III, the architecture of the MtFIP is presented. In Section IV, the inferencing performance of the MtFIP is analyzed. The last section concludes this paper.

II. Previous Works on Fuzzy Inference Processor

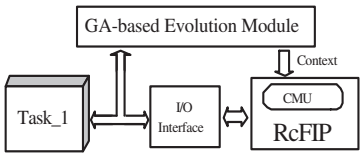
Fuzzy inference processors have been an active research area. Most fuzzy inference processors are aimed at high inference performance. Louverdis and Andreadis [12] present a digital fuzzy inference processor for image processing. A suitable architecture with a certain level of parallelism will serve to enhance inferencing performance. Pipelined architecture technique is also popular for achieving higher inferencing performance. A digital fuzzy processor with pipeline hardware architecture is proposed by Melgarejo and Pena-Reyes [13]. Aranguren *et al.* [9] present a 5-level pipeline sequential fuzzy inference processor architecture. The inferencing speed achieved is 2.4M FLIPS. Falchieri *et al.* [7] present a fuzzy inference processor involving 16 pipeline stages, with processing speed of about 6.3M FLIPS. Such fuzzy inference processors are proven to be able to offer high inferencing performance.

A context switchable digital RcFIP has been presented in [11]. The overall architecture is as shown in Fig. 1(a). There are five functional blocks in the RcFIP; fuzzy inference mapping (FIM), context management unit (CMU), address encoder module (AEM), output aggregation module (OAM) and control unit. The prototype RcFIP was implemented on Xilinx XC2V2000 Virtex-II FPGA. The FIM of the RcFIP is organized in a compact format which implicitly encodes the

Qi Cao is with Communication Systems Department, Institute for Infocomm Research, Singapore. caoqi@pmail.ntu.edu.sg
Liqin Song is with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. N060110@ntu.edu.sg
Xiaomeng Shi is with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. xmshi@ntu.edu.sg
Ju Hui Li is with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. pg01896341@ntu.edu.sg



(*a*) RcFIP Architecture



(*b*) RcFIP for a Single Task

Fig. 1. RcFIP Block Architecture

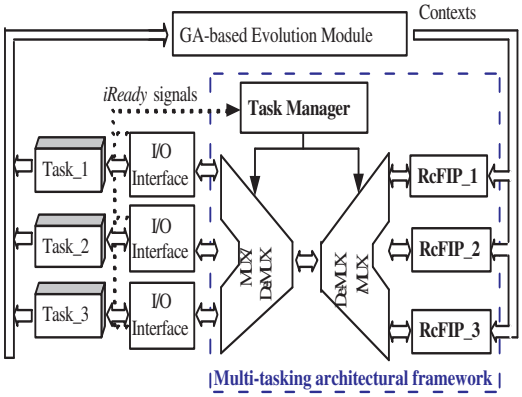


Fig. 2. Multi-Tasking Architectural Framework

fuzzy relational knowledge and the inference model. High inferencing performance of 17M FLIPS is achieved.

For conventional fuzzy inference processors, the designers specify the fuzzy contexts in a subjective manner. It results in performance that may not be optimal [14], [15], [16], [17]. In our work, the RcFIP is configured under the guidance of a genetic algorithm (GA) based evolution module, as shown in Fig. 1(*b*). Functionally, the role of the GA-based evolution module is to evolve a suitable context which is then configured into the CMU of the RcFIP [18]. The RcFIP-based system is capable of maintaining suitable level of performance in a dynamically changing operating environment. Any change in the operating domain can be accommodated by updating the context. Hence, it is possible to effect context switching without interrupting the normal operation of the RcFIP, significantly enhancing the flexibility. The proposed RcFIP-based MtFIP aims to offer both high flexibility and high inference performance.

III. Architecture of MtFIP

A. Multiple RcFIPs Architectural Framework

The concept of MtFIP is motivated by the online context switchable RcFIP with several desirable characteristics; real time reconfigurability, adaptation and high inference speed. The RcFIP is capable of online context switching for a specific task. However, it would be desirable if the fuzzy inference processor can deliver high performance for multiple tasks concurrently and allow for seamless online context switching between several tasks. This is the main objective of the proposed MtFIP.

In principle, a MtFIP can be configured to handle several tasks concurrently. In our work, the architecture of the MtFIP is illustrated by considering 3 concurrent tasks. From a functional point of view, the block architecture for executing three tasks is as shown in Fig. 2. There are three RcFIPs incorporated, each of which capable of offering good performance for a specific task. This means that each task is executed by a pre-configured task-specific RcFIP, i.e. Task_1, Task_2 and Task_3 are executed by RcFIP_1, RcFIP_2 and RcFIP_3 respectively. On the whole, high performance can be maintained, as each task is executed in a manner similar to that of a single RcFIP. High flexibility is also achieved as the framework is easily scalable to accommodate the number of tasks to be handled.

In Fig. 2, the operations of the multiplexer and de-multiplexer are handled by the Task Manager. The tasks are handled concurrently on a time-sharing basis. This can be realized through multiplexing and de-multiplexing. Data derived from each task environment are digitized by input interfaces, such as analog-to-digital converters (ADC). Digital input signals are fed selectively into the corresponding RcFIP by the multiplexer and the de-multiplexer. Each RcFIP computes the control output for a particular task. The outputs are delivered to the corresponding task environment via the multiplexer and de-multiplexer again, as well as output interfaces, such as digital-to-analog converters (DAC). The whole structure allows for great flexibility to accommodate switching among several tasks. Data from the tasks are fed to the Evolution Module as training data. Each task can be serviced by a dedicated microcontroller-based GA unit. The evolutionary process is triggered and the GA operations are carried out in the corresponding GA unit. The derived contexts with the best fitness values from the GA units are updated into the CMU of the corresponding RcFIPs. The whole setup allows for online context switching among the tasks by incorporating multiple RcFIPs according to the number of tasks to be handled.

B. Integrated Architectural Framework of MtFIP

From the multi-tasking architectural framework shown in Fig. 2, it is evident that integration of these RcFIPs on a single chip with current SoC technique will be beneficial. To achieve the integration of multiple RcFIPs, the overall architecture of the proposed MtFIP is as shown in Fig. 3. To facilitate discussions, we continue our illustration

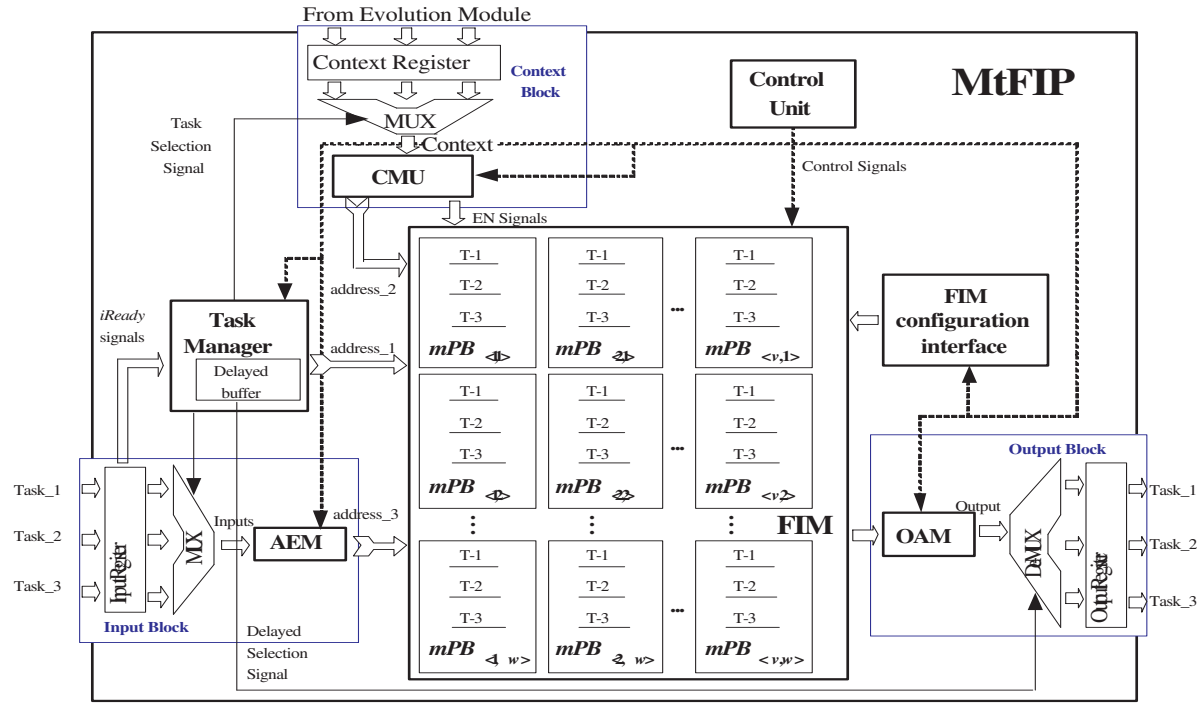


Fig. 3. Overall Architecture of MtFIP

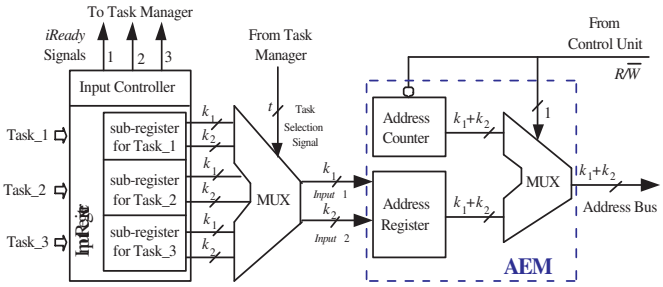


Fig. 4. Schematic of the Input Block

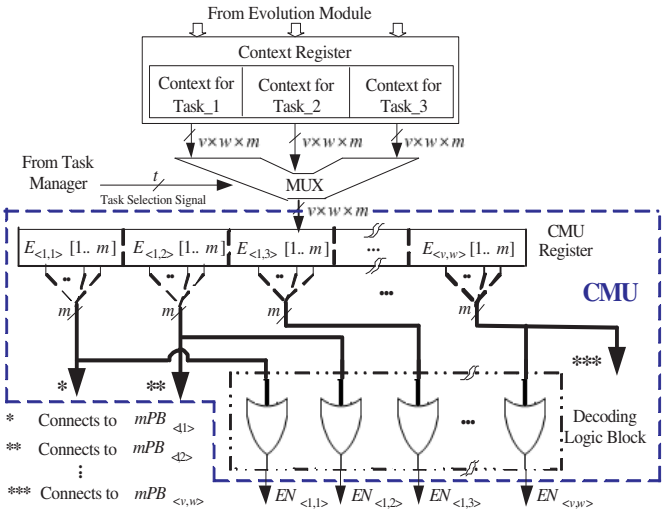


Fig. 5. Schematic of the Context Block

involving 3 tasks in describing the salient aspects of the MtFIP architecture.

The MtFIP consists of six main functional blocks as described below.

(1) FIM: It is the system's memory that maintains a compact structured representation of the fuzzy relational knowledge for multiple tasks. It is a representation that models the process of inputs fuzzification, rules evaluation and output composition. The FIM is the key part in the design of the MtFIP. It is configured specific to the tasks. The FIM is partitioned such that the data stored in the T-1 memory-space correspond to the configuration data for Task_1, while data for Task_2 and Task_3 are stored in the T-2 and T-3 memory-space respectively.

(2) Input Block: It serves as the input interface between multiple tasks and the MtFIP, which consists of an input register, a multiplexer and an AEM. It encodes input data from several tasks, for the FIM and the Task Manager. The architecture of the Input Block is shown in Fig. 4. Whenever new inputs are updated into the Input Block, the input controller pulls to high the *iReady* bit of the status register stored in the Task Manager. The *iReady* signal is important since it forms part of the information to schedule the tasks for the Task Manager.

(3) Context Block: It is the context interface between the Evolution Module and the MtFIP. As shown in Fig. 5. It consists of a context register, a multiplexer

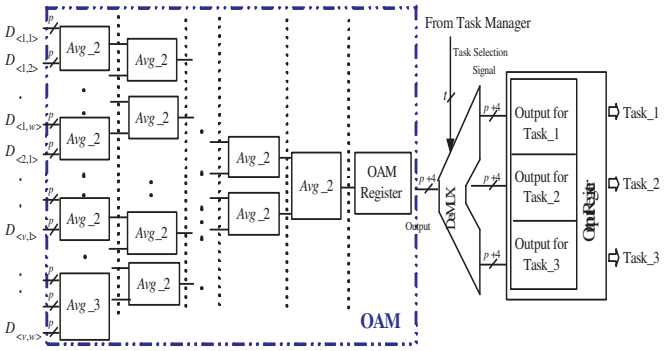


Fig. 6. Schematic of the Output Block

and a CMU. It handles context switching as well as generation of address to enable/disable the FIM partition blocks.

(4) Output Block: It provides the output interface which computes the final defuzzified output for the MtFIP and delivers the output to the corresponding task. It is made up of an OAM, a de-multiplexer and an output register, shown in Fig. 6.

(5) Task Manager: It coordinates the scheduling of the tasks. It generates address for accessing each FIM partition block. It also generates the task selection signal to manipulate the multiplexer of the Input Block and the Context Block. The de-multiplexing at the Output Block is realized using the delayed task selection signal.

(6) Control Unit: It is for the overall synchronization of the MtFIP. This includes synchronization involving the MtFIP and external hardware interface, such as ADC and the FIM configuration interface.

The architecture of AEM, CMU, OAM and control unit in the MtFIP are the same as those of the RcFIP described in [11]. The architecture of the Task Manager is newly developed for the MtFIP, which will be presented next.

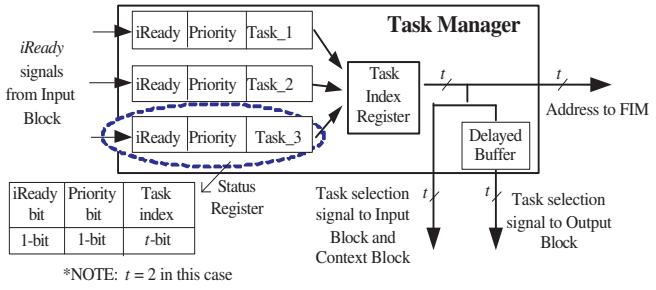


Fig. 7. Architecture of the Task Manager

1) Task Manager: The Task Manager coordinates and schedules the execution sequence of the tasks. The task selection depends not only on the setting of the *Priority* bits, but also the *iReady* bits in the status registers.

A status register is maintained for each task in the Task Manager as shown in Fig. 7. The Task Manager determines the tasks execution sequence based on information stored in the status registers. Each status register maintains a $(t+2)$-bit data. The purpose of the various status bits is explained below.

- The first bit represents the *iReady* status, indicating the readiness of inputs for a particular task. The Input Block sets the *iReady* bit to high whenever new inputs are updated into the corresponding input sub-register. It denotes that this task is ready for further execution. The *iReady* bit is cleared to '0' whenever the Task Manager finishes with the execution of this task.
- The second bit in the status register denotes the execution priority for a task. The *Priority* bits setting for all the status registers is based on a rotating priority scheme. An asserted *Priority* bit means that this task is allocated a high priority to be executed next. Among all the *Priority* bits of the status registers, only one *Priority* bit is being set to high at any time. Others are set to low denoting the lower priorities of execution. Generally speaking, the *Priority* bit of a task which is next to the newly accomplished task will be set to high. This task always gets the highest priority to be scheduled for the next round.
- The last t bits represent the index of the task. The number of bits t is equal to 2 for three tasks.

After the integrated circuit synthesis, the timing simulation results show that the Task Manager is capable of scheduling a task every two clock cycles. If the MtFIP is synthesized utilizing 66 MHz operating frequency, it means that the Task Manager schedules one task every 30 ns. Such a scheduling speed is fast enough to execute multiple tasks efficiently. In Fig. 7, the selected task's index serves as four functional control signals; as part of the address for accessing FIM partition blocks, the task selection signal for the multiplexer in the Input Block, the task selection signal for the Context Block and the delayed task selection signal for the Output Block.

IV. MtFIP Prototyping and Analysis

The VHDL prototyping of the MtFIP is carried out in Mentor Graphics FPGA Advantage 6.3 platform. After circuit simulation and synthesis, it is implemented on Xilinx Virtex-II FPGA. For the MtFIP prototype, there are 6 inputs, each being 6-bit (i.e. two inputs from each task respectively). The number of bits m to code the consequent of each rule is chosen to be 3 bits (m=3). The width of the data bus is chosen to be 8 bits (p=8). The resource utilization report is as shown in Table I.

The MtFIP deals with both inputs updated and switching of contexts. There are newly updated inputs derived from 3 tasks simultaneously. The context switching for these 3 tasks also occur at the same time. In this critical scenario, the processing time of the MtFIP will be studied. The timing simulations of this MtFIP are based on 66 MHz clock in ModelSim.

The waveforms are shown in Fig. 8. The inputs for 3 tasks are updated simultaneously at 1085 ns position (Cursor 1). The context switching for 3 tasks are effected at the same

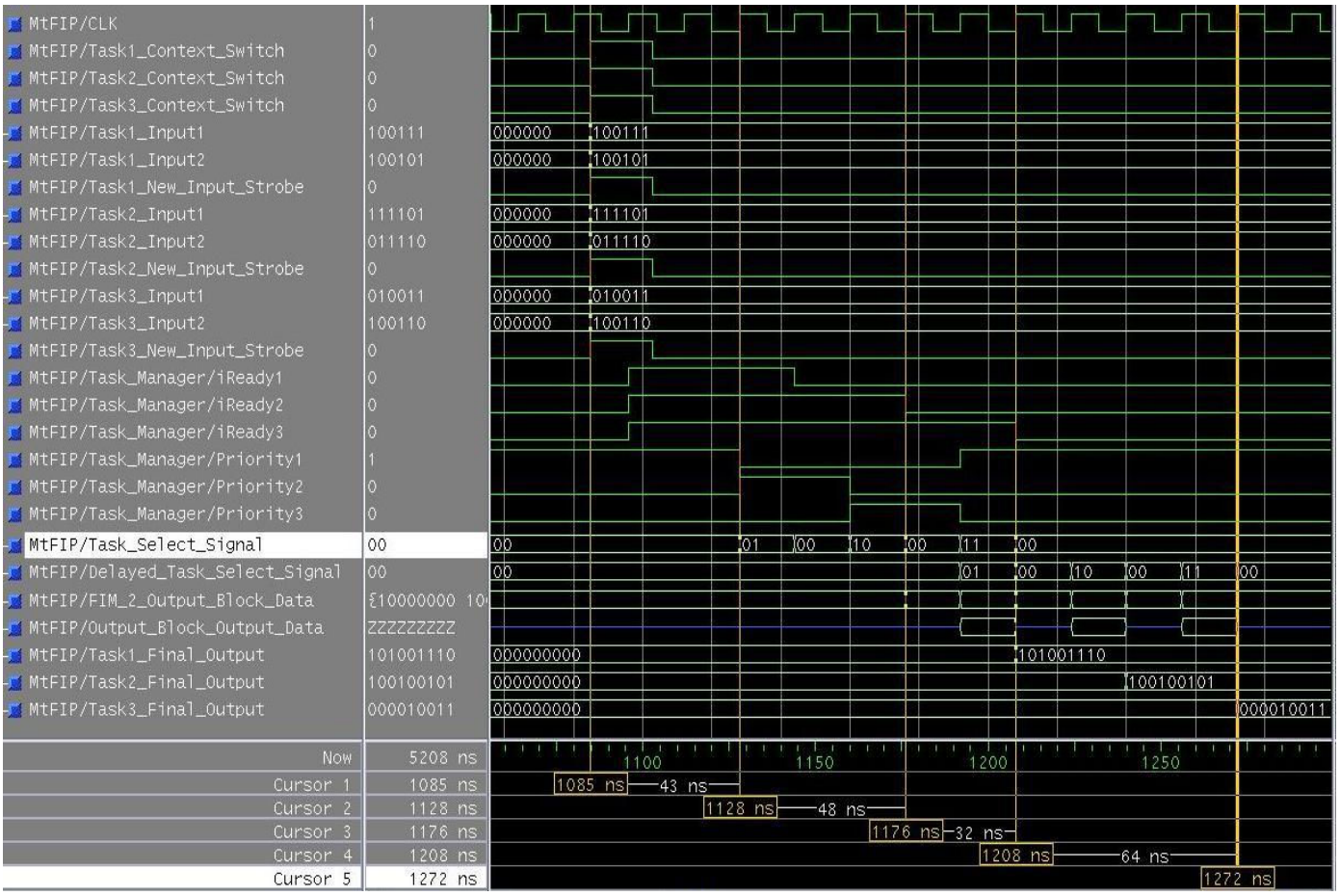


Fig. 8. Waveforms for MtFIP with both inputs and contexts updates of 3 tasks

Device Utilization for XC2V2000BG575			
Resource	Used	Avail	Utilization
IOs	289	408	70.83%
Global Buffers	7	16	43.75%
Function Generators	20689	21504	96.21%
CLB Slices	10345	10752	96.21%
Dffs or Latches	1249	22728	5.50%

TABLE I
MTFIP CIRCUITRY SCALE REPORTS

time. Within the Task Manager of the MtFIP, Task_1 has the highest priority initially. Hence, the task selection signal derived from the Task Manager is assigned to Task_1 first, indicated at 1128 ns position (Cursor 2). It is followed by Task_2 and Task_3 sequentially.

Under the control of the delayed task selection signal, the Task_1 final output is derived from the MtFIP at 1208 ns position (Cursor 4). Two clock cycles later, the Task_2 final output is derived. At 1272 ns position (Cursor 5), the Task_3 final output is derived from the MtFIP.

In this case, the first final output data for Task_1 is derived within 8 clock cycles. While the total execution time of the MtFIP for three tasks is 12 clock cycles, as shown in Fig. 8.

According to the simulation and synthesis results, the proposed MtFIP can deliver about 8.3M FLIPS (i.e. $\frac{1}{8\times 15ns}$) of inference throughput for a single task. It can provide around 5.6M FLIPS (i.e. $\frac{1}{12\times 15ns}$) for concurrent execution of 3 tasks.

The performance specifications of the proposed MtFIP are listed in Table II. It is observed that the proposed MtFIP offers good flexibility with more I/O capability and better fuzzy rules coverage. The inference speed of the MtFIP is still comparable with that of high inference performance fuzzy processors reported in [11], [19], [20], [5], [21], [22].

As the counterpart of the proposed MtFIP, multiple processors in a single chip approach has the outstanding advantage with true parallel executions for multiple tasks concurrently. Each processor can provide all its resources to execute a single task. High throughput and reliability can be achieved. The communications between each processor and the corresponding task are quite straightforward. However, the communications and resource sharing among the processors are not so efficient. Usually there is a master processor or host to coordinate the information sharing among the processors.

	MtFIP	[11] RcFIP	[19] 2007	[20] 2006	[5] 2006	[21] 2006	[22] 2005
Processor Prototype	Xilinx VirtexII	Xilinx VirtexII	Xilinx Spartan-3	Xilinx Spartan-3	Altera APEX20K	Micro-controller	ASIC
Number of Inputs	6	2	2	2	2	2	2
Number of Outputs	3	1	1	1	1	1	1
Number of Clock Cycles per Logic Inference	8 (single task) 12 (multi tasks)	4	9	11	–	–	–
Inference Performance (Mega FLIPS)	8.3 (single task) 5.6 (multi tasks)	17	–	15	–	–	7.0
Online Reconfigurability	Yes	Yes	No	No	No	No	No
Multi-tasking Capability	Yes	No	No	No	No	No	No

TABLE II
MTFIP SPECIFICATIONS AND COMPARISONS

While the MtFIP facilitates resource sharing. The potential tradeoff of the MtFIP is that the performance may not be as good as that of multiple processors in the single chip approach, since MtFIP executes multiple tasks on a time-sharing basis.

Although under time-sharing execution basis, the MtFIP is able to execute a single task request within 8 clock cycles (i.e. 120 *ns* under 66 MHz working frequency), and execute 3 concurrent requests within 12 clock cycles (i.e. 180 *ns* under 66 MHz working frequency). These speeds are much faster than those of the Input/Output peripheral between the MtFIP and domain applications. Hence, from the domain applications' viewpoint, the time-sharing execution of the MtFIP does not affect the overall performance of the whole systems.

V. CONCLUSION

A fuzzy inference processor with multi-tasking capability is proposed in this paper. The specifically configured architecture of the MtFIP can achieve similar inferencing performance compared to that of a RcFIP executing one task. The MtFIP endows the chip with the capacity to handle several tasks concurrently. It has a compact structure as a result of sharing hardware resources. The multiple RcFIP architecture share the same Input Block, Context Block, Output Block, Task Manager and Control Unit, except for the FIM. The FIM covers the specific configuration data for these tasks in its partition blocks. The switching of task takes effect when the address signals accessing the FIM in the MtFIP are changed.

The MtFIP is more versatile in handling context independent inferencing both within the same task and among several tasks. The implication of context independence is that it allows the chip to be programmed to handle any situation pertaining to the tasks without the need for offline reprogramming. With online context switching, situations where the task scenarios change dramatically can be handled seamlessly. This distinct feature of the MtFIP allows context switching among multiple tasks during the whole procedure, without noticeable degradation in performance compared to a task-dedicated inference processing scheme.

REFERENCES

[1] W. O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SoC platforms: A component-based design approach," *IEEE Des. Test Comput.*, vol. 19, no. 6, pp. 52–63, Nov 2002.

[2] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard, "Application of a multi-processor SoC platform to high-speed packet forwarding," in *Design, Automation & Test in Europe Conf.*, pp. 58–63, Feb 2004.

[3] B. D. Theelen, A. C. Verschueren, V. V. R. Surez, M. P. J. Stevens, and A. Nu, "A scalable single-chip multi-processor architecture with on-chip RTOS kernel," *J. of Syst. Architect.*, vol. 49, no. 12-15, pp. 619–639, Dec 2003.

[4] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van de Wolf, O. P. Gangwal, A. Timmer, and E. J. D. Pol, "A heterogeneous multiprocessor architecture for flexible media processing," *IEEE Des. Test Comput.*, vol. 19, no. 4, pp. 39–50, Jul 2002.

[5] M. S. Masmoudi, I. Song, F. Karray, M. Masmoudi, and N. Derbel, "Hardware/software approache for the FPGA implementation of a fuzzy logic controller," in *Int. Conf. Design & Test of Integrated Syst. in Nanoscale Tech.*, pp. 419–423, Sep 2006.

[6] M. H. Lim and Y. Takefuji, "Implementing fuzzy rule-based systems on silicon chips," *IEEE Expert*, vol. 21, no. 3, pp. 31–45, Feb 1990.

[7] D. Falchieri, A. Gabrielli, and E. Gandolfi, "Very fast rate 2-input fuzzy processor for high energy physics," *Fuzzy Sets and Syst.*, vol. 132, no. 2, pp. 261–272, Dec 2002.

[8] M. H. Lim, J. H. Li, and Q. Cao, "On-line evolving fuzzy hardware system for atm cell-scheduling," *J. of Syst. Architect.*, vol. 52, no. 3, pp. 169–183, Jul 2006.

[9] G. Aranguren, L. A. L. Nozal, X. Basogain, J. L. Martín, and J. L. Arroyabe, "Hardware implementation of a pipeline fuzzy controller and software tools," *Fuzzy Sets and Syst.*, vol. 128, no. 1, pp. 61–79, May 2002.

[10] M. H. Lim, Q. Cao, and J. H. Li, "Evolvable hardware using context switchable fuzzy inference processor," *IEE Proc. - Comput. Digit. Tech.*, vol. 151, no. 3, pp. 301–311, Jul 2004.

[11] Q. Cao, M. H. Lim, J. H. Li, Y. S. Ong, and W. L. Ng, "A context switchable fuzzy inference chip," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 4, pp. 552–567, Aug 2006.

[12] G. Louverdis and I. Andreadis, "Design and implementation of a fuzzy hardware structure for morphological color image processing," *IEEE Trans. Circ. Syst. Video Tech.*, vol. 13, no. 3, pp. 277–288, Mar 2003.

[13] M. Melgarejo and C. A. Pena-Reyes, "Implementing interval type-2 fuzzy processors," *IEEE Comput. Intell. Mag.*, vol. 2, no. 1, pp. 63–71, Feb 2007.

[14] D. Kim, "A design of CMAC-based fuzzy logic controller with fast learning and accurate approximation," *Fuzzy Sets and Syst.*, vol. 125, no. 1, pp. 93–104, Jan 2002.

[15] M. H. Lim, S. Rahardja, and B. H. Gwee, "A ga paradigm for learning fuzzy rules," *Fuzzy Sets and Syst.*, vol. 82, no. 3, pp. 177–186, Sep 1996.

[16] M. H. Lim and W. Ng, "Iterative genetic algorithm for learning efficient fuzzy rule set," *Artif. Intell. for Eng. Design Analysis & Manufac.*, vol. 17, no. 3, pp. 335–347, Sep 2003.

[17] M. H. Lim, Y. Yu, and S. Omatu, "Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem," *Comput. Optim. Appl.*, vol. 15, no. 3, pp. 249–268, Jul 2000.

[18] J. H. Li, M. H. Lim, and Q. Cao, "A qos-tunable scheme for atm cell scheduling using evolutionary fuzzy systems," *Applied Intell.*, vol. 23, no. 3, pp. 207–218, Sep 2005.

[19] J. L. González, O. Castillo, and L. T. Aguilar, "FPGA as a tool for implementing non-fixed structure fuzzy logic controller," in *IEEE Symp. Foundations of Comput. Intell.*, pp. 523–570, Apr 2007.

[20] K. M. Deliparaschos, F. I. Nenedakis, and S. G. Tzafestas, "Design and implementation of a fast digital fuzzy logic controller using FPGA technology," *J. Intell. Robot. Syst.*, vol. 45, no. 1, pp. 77–96, Jan 2006.

[21] A. R. Ofoli and A. Rubaai, "Real-time implementation of a fuzzy logic controller for switch-mode power-stage DC-DC converters," *IEEE Trans. Ind. Appl.*, vol. 42, no. 6, pp. 1367–1374, Nov/Dec 2006.

[22] S.-H. Huang and J.-Y. Lai, "A high-speed VLSI fuzzy inference processor for trapezoid-shaped membership functions," *J. Info. Sci. Engi.*, vol. 21, no. 3, pp. 607–626, May 2005.