An FPGA-based Distributed Computing System with Power and Thermal Management Capabilities

Hao Shen, Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, NY, 13902, USA

ABSTRACT

Runtime power and thermal management has attracted substantial interests in multi-core distributed embedded systems. Fast performance evaluation is an essential step in the research of distributed power and thermal management. Compared to software simulation, an FPGA-based evaluation platform provides fast emulation speed which enables us to test the performance of power/thermal management policies with real-life applications and OS. Compared to computer clusters, an FPGA-based platform has the flexibility to be configured into any network topology and hardware sniffer for performance monitoring can be added easily. This paper presents an FPGA based emulator of multi-core distributed embedded system designed to support the research in runtime power/thermal management. The system consists of multiple FPGAs connecting through Ethernet with each FPGA configured as a multi-core system. Hardware and software supports are provided to carry out basic power/thermal management actions including inter-core or inter-FPGA communications, runtime temperature monitoring and dynamic frequency scaling.

Keywords

FPGA, multi-core distributed system, power management, thermal management, DVFS

1. Introduction

High power consumption and high working temperature have become a major issue in the design of today's embedded systems. They increase cooling cost, degrade the system reliability and also reduce the battery cycle time in portable devices. Recently, research in Dynamic Power Management (DPM) and Dynamic Thermal Management (DTM) has attracted substantial interests. The DPM controller dynamically slows down or turns off the computing device when it is idle or under-utilized to reduce the system power consumption [21]. The similar actions are taken by the DTM controller when the die is (or is predicted to be) over heated [22]. The effectiveness of DTM and DPM relies heavily on the workload pattern of the computing device, which is determined by task scheduling and ordering policies, the nature of applications running in the system as well as user input activities. Furthermore, the impact from the OS, such as the overhead of context switch and power mode switch, also affects the efficiency of DTM and DPM.

In this paper, we present an FPGA-based distributed computing platform designed to support the research in distributed embedded computing with power/thermal management capabilities. The system consists of multiple FPGAs connecting through Ethernet links with each FPGA configured as a multi-core system. Hardware and software supports are provided to carry out basic power/thermal management actions including inter-core and inter-FPGA communications, runtime temperature monitoring and dynamic frequency scaling. In the experiments, we evaluated the inter-FPGA and inter-core communication delay, tested the function of the distributed computing system using a case study of parallel matrix multiplication, and evaluated the function of dynamic frequency scaling and temperature sensing by measuring the temperature change of the CPU when it is running at different frequencies.

Compared to software simulation, an FPGA-based evaluation platform provides fast emulation speed which enables us to test the performance of power/thermal management techniques with real-life applications and OS. Compared to computer clusters, an FPGA-based platform has the flexibility to be configured to any hardware architecture and network topology. Hardware based sniffers for performance monitoring and traffic analysis can also easily be added.

Many works have been performed to emulate or develop multiprocessor distributed system using FPGA. Reference [1] gives a good overview of FPGA-based multiprocessor systems. References [3][4] aim at creating an FPGA-based multiprocessor or distributed platforms for teaching purpose. References [5][6] focus on the memory and synchronization problems in an FPGA-based multi-processor system. The authors of [7][8][9] propose different architectures for FPGA-based multiprocessor or distributed system for video applications.

There are also works focusing on FPGA-based multi-processor or distributed system emulator for power and thermal optimization. For example, the authors of [10] present a hardware and software co-systhesis framework for FPGA-based distributed embedded system in order to achieve low power design. Reference [11] uses FPGA to emulate the power consumption of a multiprocessor system in order to guide task migration. Reference [2] presents the HW/SW of an FPGA-based emulation framework that enables the rapid extraction of a large range of statistics, including thermal modeling , at different architectural levels of MPSoC designs. Both [11] and [2] demonstrate that the FPGA-based HW/SW emulation achieves much faster speed than the software simulation.

Our work differs from all the previous works as we created a generic distributed platform using FPGA where power and thermal managements are possible. The main contributions of this paper are:

1. We created a multi-core distributed computing platform using *Nios II Embedded Evaluation Kit* (*NEEK*). The platform consists of multiple FPGAs with each FPGA configured as a multi-core system. The processors on the same FPGA communicate with each other through shared memory and different FPGAs communicate with each other through Ethernet links using the TCP/IP stack.
2. Each core is on a separate clock domain and has the dynamic frequency scaling capability.
3. Each core has its own temperature sensor that mapped to its local address space.
4. Software supports for inter-core and inter-FPGA communication, dynamic frequency scaling and temperature monitoring are provided. Their latency and overhead are analyzed.

*This work is supported in part by NSF under grant CNS-0845947

978-1-4577-0638-7 /11/$26.00 ©2011 IEEE

5. A case study is provided that shows how to do distributed matrix multiplication using the proposed platform.

The rest of the paper is organized as follows: Section 2 briefly introduces the software and hardware resources on Altera Nios II Embedded Evaluation Kit, which is the base device of our distributed computing platform. Section 3 presents the details of the system architecture and Section 4 talks about the design of frequency scaling and temperature monitoring modules. Section 5 gives the experimental results and analysis. Section 6 is the conclusion.

2. Altera Nios II Embedded Evaluation Kit (NEEK)

Although the proposed distributed computing platform can be implemented on any major FPGA, its architecture selection is more or less affected by the hardware and software resources on the FPGA that is chosen for this project. In this section, we give a brief introduction of the resources on the Altera Nios II Embedded Evaluation Kit (NEEK). We will only focus on those resources that will be used in our work.

The Altera NEEK Cyclone III edition has one Cyclone III EP3C25F324 FPGA with 25,000 Logic Elements and 594 Kbits embedded memory, 32 MB DDR SDRAM, 1MB SRAM, 16 MB Intel P30/P33 Flash, 800 X 480 touch-screen LCD, Ethernet 10/100 Mbps, and PS2 and RS-232 connector,. The Nios II is a soft IP core of the embedded processor designed specifically for Altera FPGA. In this work, we choose the MicroC/OS-II RTOS [12] whose full ANSI C source code is included in the Nios II Embedded Design Suite. The MicroC/OS-II is a multitasking operating system that supports maximally 64 tasks with distinct task priorities (ranging from 1 to 64). It supports preemptive scheduling, and always runs the highest priority task that is ready.

The full ANSI C source code of NicheStack TCP/IP Network Stack (Nios II Edition) is also distributed by Altera as part of the design suite. The network stack can be used with the 10/100 Mbps Ethernet Controller (PHY) provided by Altera to establish socket communication via Ethernet links.

3. System Architecture

The proposed distributed computing platform has hie-rarchical architecture, which is shown in Figure 1. The system consists of multiple FPGAs connecting via Ethernet links. Each FPGA is further configured as a multi-core system. The cores on the same FPGA communicate via shared memory, while different FPGAs communicate via Ethernet Links.

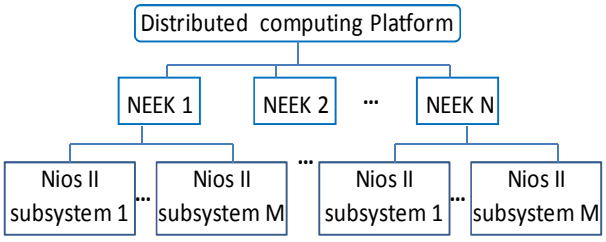


Figure 1. Hierarchical architecture

3.1 Single FPGA multi-core system

There are multiple Nios II subsystems on a single FPGA.

Figure 2 shows the block diagram of one of the Nios II subsystem including its memory and some basic peripherals. The JTAG UART provides the debugging port interface. A High Resolution Timer is included to measure the program execution time. The *parallel I/O* (*PIO*) provides control and monitoring to clock generation block and the temperature sensor, which will be discussed in Section 4.

One or multiple shared memories are connected to each Nios II subsystem. These shared memories are configured as hardware mailbox for communications among processors on the same FPGA. Among all the Nios II subsystems on the same FPGA, there is one that has Ethernet interface module. The Ethernet interface module consists of a DMA Controller, a Descriptor Memory and the Ethernet MAC. The Nios II subsystem that has the Ethernet interface acts as a gateway for inter-FPGA communications in the distributed computing system. All peripherals are connected to the Nios II processor via Avalon memory mapped interface [23]. The Avalon streaming interface [23] is used to connect the DMA to the Ethernet controller.

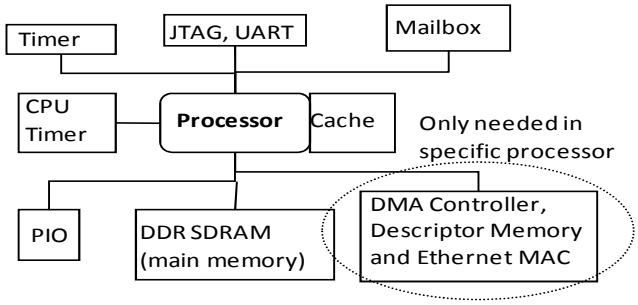
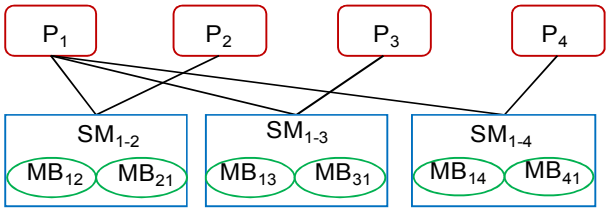


Figure 2. Processor configuration

The on-chip inter-processor communication is achieved using the mailbox [13]. Figure 3 shows an example of the topology of connections among 4 on-chip processors. Between any two processors that communicate to each other, a shared memory is inserted. The shared memory is configured into 2 mailboxes. One of them stores outgoing messages and the other stores incoming messages. The mailbox core contains hardware mutexes to ensure the mutually exclusiveness during communication through the shared memory where the actual messages are stored. In Figure 3, MB_{12} is used to store the messages sent from P_1 to P_2 while MB_{21} is for those sent from P_2 to P_1. Both MB_{12} and MB_{21} are attached to the same on-chip memory block that connects to both processors. Note that Figure 3 shows only a simplified system where the communication can only happen between P_1 and other 3 processors. The system can be configured to implement different topologies of inter-processor communication network, such as a ring or a mesh.



P:processor; SM: on-chip shared memory; MB: Mailbox

Figure 3. Topology of on-chip inter-processor connections

To post a message to an outgoing mailbox, the Mailbox API function altera_avalon_mailbox_post() must be used. The Mailbox API function altera_avalon_mailbox_pend() can be used to read a message from an incoming mailbox. The altera_avalon_mailbox_pend() function provides blocking wait for a message in the specified mailbox. In a multi-tasking OS such as MicroC/OS-II, if the mailbox is empty when the function is called, the process will be blocked and switched to be inactive. The process will be switched to the ready list when the mailbox is no longer empty.

It is necessary to point out that although the mailbox core and its supporting API provide a convenient way to achieve inter-processor communication, its speed is relatively slow for large data communication as we will show in the experimental results. This is because the mailbox API allows the processor to read or write only one 32 bit message each time. For high speed inter-processor communication of large amount of data, hardware mutex core [14] with shared memory should be used.

3.2 Multi-FPGA distributed system

The overall distributed computing platform consists of multiple FPGAs connecting via Ethernet links. We use the notation K_iP_j to represents the jth processor on the ith NEEK. All NEEKs are connected through their Ethernet port, which is connected to the first processor of the kit. Therefore, the K_iP_1, $1 \le i \le 4$ acts as the communication gateway. The gateway processor establishes the socket TCP/IP connections with all other gateway processor in the distributed system. For each TCP/IP socket, a receiving process is created that performs blocking wait for the incoming data. To send data from K_iP_x to K_jP_y, where $i \neq j$ and $x, y \neq 1$, the processor K_iP_x first sends the data to the gateway processor K_iP_1 via the mailbox, the data is then transmitted from K_iP_1 to K_jP_1 through TCP/IP socket, and finally the data is sent from K_jP_1 to K_jP_y via the mailbox.

As we can see from the above analysis that, in addition to the user applications, each processor must maintain a set of receiving processes waiting for the incoming data from either the mailboxes or the socket interfaces. We use semaphores to coordinate the execution of different processes. For example, in the matrix multiplication example that will be discussed in Section 5.2, the calculation process will wait on a semaphore which will be released by the receiving process when all data for the calculation are ready.

4. Frequency Scaling and Temperature Monitoring

Each Nios II subsystem has a hardware clock generation and selection block and a temperature sensor that provides the processor with dynamic frequency scaling and temperature monitoring capabilities.

4.1 Glitch free clock switching

One way to achieve dynamic clock frequency scaling is to reconfigure the *Phase-Locked Loop* (*PLL*) during the runtime. The Altera Phase-Locked Loop Reconfiguration Megafunction [16] can be used for this purpose. However, to reconfigure the Altera PLL [17] during runtime takes several microseconds, which is quite large performance overhead. Furthermore, the Reconfiguration Megafunction consumes lots of reconfigurable logic resources as well as on-chip memories, and it requires complicated control.

The second way to achieve dynamic frequency scaling is to switch among a set of clocks running at discrete frequency levels. Each Altera PLL generates up to 5 different output clocks, which is the same as the number of frequency levels supported by an XScale processor. The biggest challenge in designing a clock selection module is how to avoid the clock glitch during the transition period.

Figure 4 shows the glitch free clock generation and selection module implemented in our system. Based on the 4 bit control signal (i.e. *control*[3:0]) , it performs 4-to-1 selection from the output clocks generated by the PLL. In this design, the ALTPLL block is the Altera PLL Megafunction; the ALTCLKCTRL block is the Altera Clock Control Megafunction which performs clock enable/disable and clock selection [18]; and the LPM_MUX is a normal multiplexer. The DFF0~DFF3 are D-flip flops. They introduce delays during certain clock transitions to prevent glitch. More details about their function will be discussed later.

Note that the ALTCLKCTRL can only performs 2-to-1 clock selection, if the clock signals are generated by the PLL. Therefore, we have to used 2 ALTCLKCTRL blocks to multiplex 4 clock signals generated by PLL. To guarantee glitch free clock transition, the "ensure glitch-free switchover implementation" option of the ALTCLKCTRL must be enabled. At anytime, at most one ALTCLKCTRL is active.

The Quartus II software does not support concatenation of clock controllers. That is why a normal multiplexer is used to select the outputs of the 2 ALTCLKCTRL blocks. In Quartus II Analysis & Synthesis settings, we enable the "Clock Mux Protection" option. This option ensures that the multiplexers in the clock network to be decomposed to 2-to-1 multiplexers, each of which will be synthesized to one LUT and hence be glitch free [15].

A clock generation/selection module is associated to each Nios II subsystem. Its control signals (*control*[3:0])are connected to the parallel I/O of the Nios II subsystem. The processor changes its frequency by writing to the corresponding parallel I/O ports. Because at most one ATLCLKCTRL is enabled at anytime, the value of control[3] and control[2] cannot be 1 at the same time.

Figure 5 illustrates the simulated waveform of an example of clock switching. Assume that the PLL clocks $c0$, $c1$, $c2$ and $c3$ are running at frequency 25MHz, 50MHz , 75 MHz , and 100MHz. The input clock of PLL is 50MHz. Also assume that the control[3:0] is initially "0100". The ALTCLKCTR1 controller is disabled and gives constantly low output while the ALTCLKCTR0 controller is enabled.

The output clock is connected to PLL output $c0$ through LPM_MUX and ALTCLKCTR0. After the processor write "0110" to control[3:0], the ALTCLKCTRL switches from $c0$ to $c1$. There is no glitch in the output clock because the switching occurs in the Altera glitch free clock control block. In the next, the processor writes "1001" to control[3:0]. This command will eventually turn off the ALTCLKCTR0 and switch the LPM_MUX from ALTCLKCTR0 to ALTCLKCTR1. Although the LPM_MUX made the switch immediately after the control signal changed its value, the ALTCLKCTR1 is still disabled at this time because its enable signal is delayed by 2 clock cycles. The output clock will stop for a very short period of time during transition and resume afterwards. In this way, glitch on the clock is prevented.

The waveform shows that our clock selection unit takes only about 50 nanoseconds for clock transition. Compare to the PLL Reconfiguration Megafunction which has several microseconds delay, the clock selection unit has much lower performance overhead and needs much less hardware resource.

Since the processor's frequency will be changed at runtime and other components (e.g., memory and other peripherals) work under a fixed clock frequency, we need to bridge the gaps between processor's clock domain and other components clock domain. Avalon Memory Mapped Clock Crossing Bridge [19] is inserted between the processor and memory while Avalon Memory Mapped Pipeline Bridge [19] is inserted between the processor and peripherals.

4.2 Monitor the processor's temperature

In order to provide our system with the thermal awareness, an on-chip temperature sensor built up-on the programmable logic resources are implemented and attached to each Nios II subsystem.

We adopted the architecture of the temperature sensor described in [20]. The temperature sensor uses a delay line whose delay increases with the increasing of the temperature around it so that the delay time can be a good measurement of the temperature. In order to monitor the temperature change of a processor, we implemented this sensor

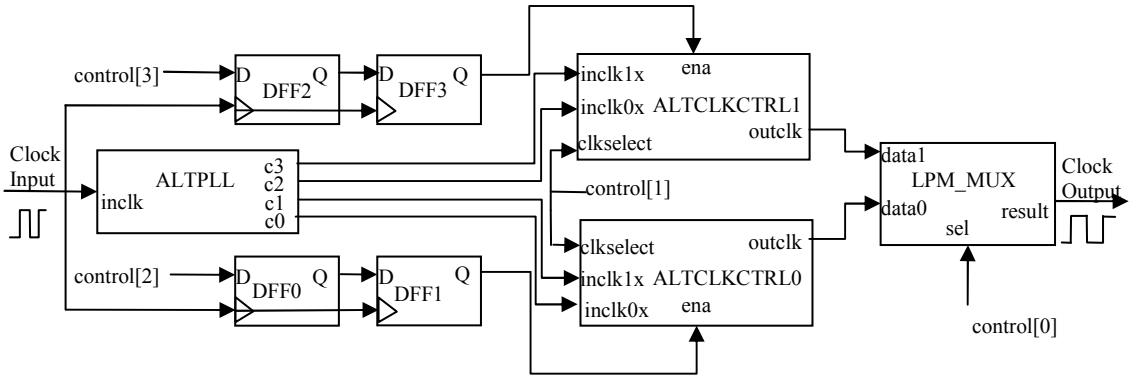


Figure 4.Clock generation block

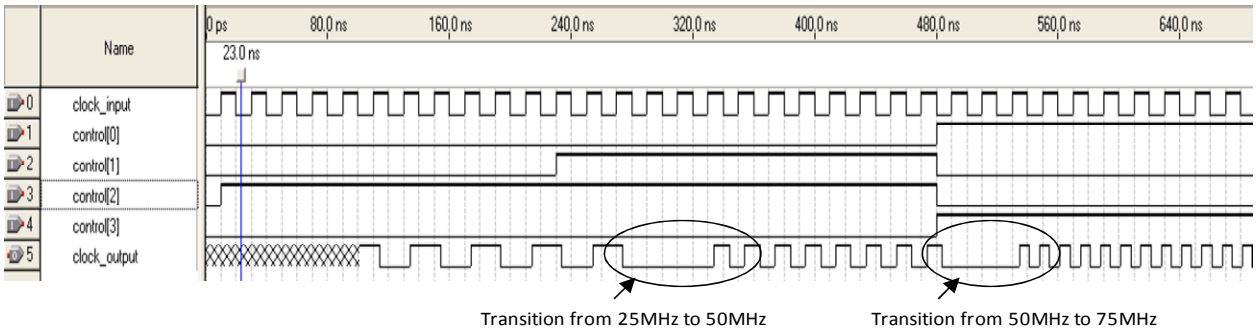


Figure 5. Simulation result of clock transition proces

and put it inside the corresponding processor using the chip planar in Quartus II. The resource used for the sensor is very little. Only a little more than 200 logic elements are needed to implement the temperature sensor in our system. The simplified schematic of the sensor is shown in Figure 6**.** After writing to the 'START' signal at the very beginning, the processor

can continuously read the temperature value from the sensor via the PIO interface.

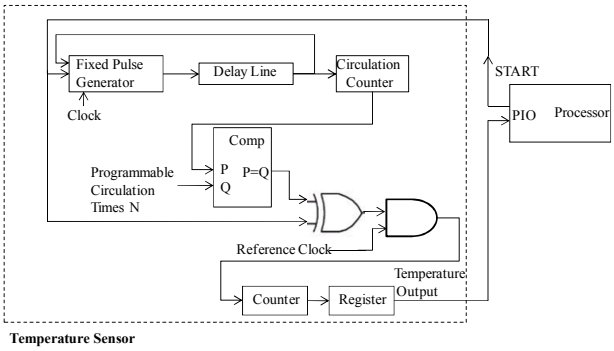


Figure 6. Temperature sensor and processor

5. Experiments

We used Quartus II and SOPC builder to create the hardware prototype configured in the FPGA and then used the Nios II Software Build Tools (SBT) to develop the user applications and to port them with the hardware abstraction layer (HAL) and operating system to the FPGA board. Our distributed computing platform consists of 4 NEEKs and each NEEK is configured into 4 Nios II subsystems. The 4 Nios II processors have point-to-point communication between each other and each mailbox between 2 processors is 256 byte large. By default, the CPUs are running at 66.5 MHz. Standard Nios II cores are used with 1 KB instruction cache.

Three experiments have been carried out. The first experiment characterizes the latencies of inter-FPGA communication and inter-processor communications. The second experiment verifies the function of the distributed computing platform using a parallel matrix multiplication program. Finally, the third experiment evaluates the function of the clock selection module and the temperature sensor, by measuring core temperature while changing the core clock frequency.

5.1 Characterization of communication latency

Communication latency is an important characteristic of a distributed computing system. This information is usually required by task scheduling and mapping algorithms for power and performance optimization. In the first experiment, we measured the latency of inter-FPGA and inter-processor communications.

The communication latency between K_iP_x and K_jP_y ($1 \le i, j \le 4$ and $1 \le x, y \le 4$) is defined as the duration starting from the time when K_iP_x begins to send the data till the time when K_jP_y receives all the data. K_iP_x and K_jP_y are asynchronous to each other. We cannot find a common time reference to measure the time between events on these two systems. Therefore, once K_jP_y receives the packet, it will immediately send back the same packet to K_iP_x. We measure the round trip latency of the packet and divide it by 2 to calculate the one-way latency. This measurement is applicable because our system is homogenous and symmetrical.

Figure 7 shows the communication latency between K_iP_1 and K_jP_1, where $1 \le i, j \le 4$ and $i \ne j$. As both processors are gateway on different NEEKs, this communication involves only the Ethernet Links. As we can see that the latency is almost a linear function to the transmission size. It takes about 8 milliseconds to send 5 KB data.

Figure 8 shows the communication latency between K_iP_x and K_iP_y, where $1 \le i \le 4$, $1 \le x, y \le 4$ and $x \ne y$. Both of the processors are on the same FPGA and their communication involves only the mailbox links. Again, the communication latency is a linear function of the transmission size. As we mentioned earlier, the mailbox based communication is relatively slow for large size of data because each time only one 32 bit message can be sent or received.

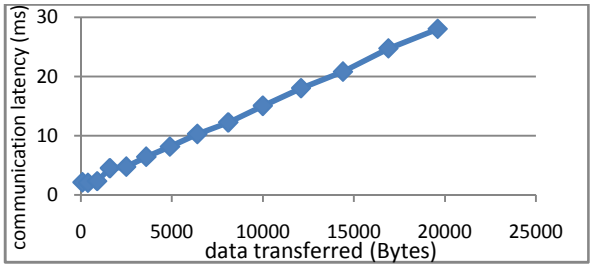


Figure 7. Inter-FPGA communication latency

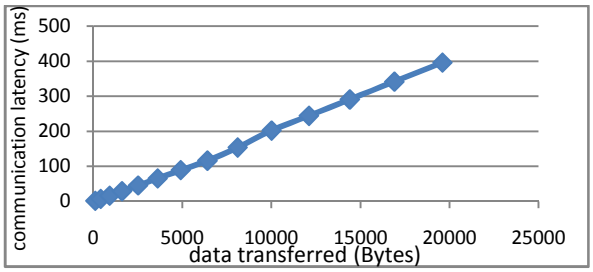


Figure 8. Inter-processor communication latency

The communication latency between non-gateway pro-cessors on different FPGAs consists of the time spent on the 2 mailbox links and one Ethernet links. For example, if we want to transfer 4KB data from K_2P_2 to K_1P_2, from Figure 7 and Figure 8 , the overall transfer time will be 2*(transferring 4KB data time through mailbox)+1*(transferring 4KB data time through Ethernet) ≈ (2*70.5+6.9) = 147.9 ms. The measured communication latency is roughly 148.3 ms which is very close to our estimation.

5.2 Parallel matrix multiplication

In this experiment we test the function of the distributed computing platform using a parallel matrix multiplication program.

Given three NxN matrices $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$. Let $\boldsymbol{C} = \boldsymbol{AB}$. If we equally divided each matrix into M rows and M columns, then we create MxM sub-matrices from the original matrix. Denote those sub-matrix using their row and column index, we will have matrices, $\boldsymbol{A}_{i,j}$, $\boldsymbol{B}_{i,j}$ and $\boldsymbol{C}_{i,j}$, $1 \le i, j \le M$. It can be proved that $\boldsymbol{C}_{x,y} = \sum_{i=1}^{M} \boldsymbol{A}_{x,i}\boldsymbol{B}_{i,y}$. The matrix multiplication can be rewritten as the following:

$$\boldsymbol{C} = \boldsymbol{AB} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \cdots & \boldsymbol{A}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{M,1} & \cdots & \boldsymbol{A}_{M,M} \end{bmatrix} \begin{bmatrix} \boldsymbol{B}_{1,1} & \cdots & \boldsymbol{B}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{B}_{M,1} & \cdots & \boldsymbol{B}_{M,M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{1,1} & \cdots & \boldsymbol{C}_{1,M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{C}_{M,1} & \cdots & \boldsymbol{C}_{M,M} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{M} \boldsymbol{A}_{1,i}\boldsymbol{B}_{i,1} & \cdots & \sum_{i=1}^{M} \boldsymbol{A}_{1,i}\boldsymbol{B}_{i,M} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{M} \boldsymbol{A}_{M,i}\boldsymbol{B}_{i,1} & \cdots & \sum_{i=1}^{M} \boldsymbol{A}_{M,i}\boldsymbol{B}_{i,M} \end{bmatrix}$$

Based on such decomposition method, we designed the parallel matrix multiplication program. The matrix size N is 100 in our experiment and M is set to 1, 2, 3, and 4. The matrix is initially stored in K_1P_1. After establishing the TCP/IP socket, K_1P_1 sends the sub-matrices $A_{x,1\sim M}$ and $B_{1\sim M,y}$, $1 \le x, y \le M$, to processors either on the same FPGA or on a different FPGA. Upon receiving the data, each processor then performs the calculation $\boldsymbol{C}_{x,y} = \sum_{i=1}^{M} \boldsymbol{A}_{x,i}\boldsymbol{B}_{i,y}$ and

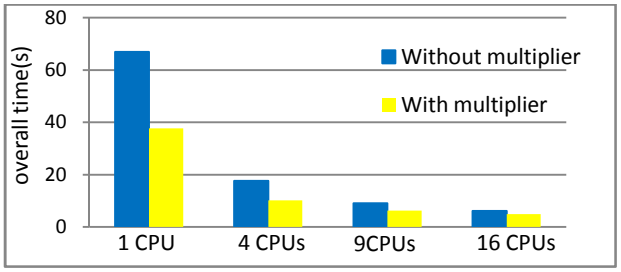


Figure 9. Overall task execution time for processors with and without embedded multipliers

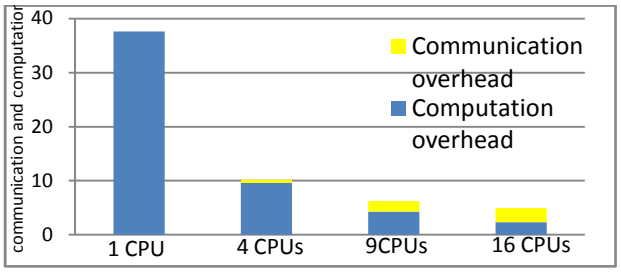


Figure 10. Computation and communication time for processors with embedded multipliers

sends the result matrix $\boldsymbol{C}_{x,y}$ back to K_1P_1. Based on the algorithm, the computation is distributed onto M^2 processors.

Figure 9 shows the overall task execution time of those 4 scenarios where M varies from 1 to 4 and the number of CPUs involved in the computing varies from 1 to 16. The execution time for processors with or without hardware multipliers are both presented. Figure 10 shows the communication and computation time break down for processors with embedded multipliers. As we can see, with the processor number increasing, the overall task execution time decreases almost linearly. The embedded multipliers can significantly reduce the execution time because the computation accounts for the major part of the execution time. And when we increase the number of processors involved in the computation, the com-munication time increases and becomes comparable to the computation time since more data need to be transferred while every processor performs less computation.

5.3 Evaluation of the temperature sensor and clock selection module

In the third experiment, we dynamically switches the processor's working frequency from 1MHz to 10MHz, 50 MHz , and 100MHz and evaluate the thermal impact of the frequency scaling. During the entire experiment, the processor is executing the floating point matrix multiplication program discussed in previous subsection. A separate process is created which reads the temperature sensor every 2 minutes. The processor stays at each frequency level for 10 minutes. Therefore, 5 temperature data are collected for each clock frequency levels.

When all frequency levels have been tested, the processor stays idle at 100MHz for 10 minutes and go back to execute the matrix multiplication program at 1MHz. And the previous procedure repeats.

Figure 11 shows the trace of temperature changes in the above mentioned procedure. As we can see, the reading of the temperature sensor increases when the processor is running at a higher frequency. And at the same time, when processor is idle, thought the working

frequency doesn't change, the temperature goes down as we expected. However, because the Nios II subsystem is not clock gated, even though no instruction is executed in the processor, the clock is still toggling and there is still switching activities. Therefore, the temperature of the idling 100 MHz processor is still higher than the temperature of an active 10 MHz processor.

In average, the temperature sensor reading is 8670 when the CPU is running at 1MHz and 8730 when the CPU is running at 100 MHz. As in Figure 6, we use 150 logic elements to build the delay line and the circulation times is 4096. Based on this and according to [20], 1 unit difference in sensor reading corresponds to about 0.03°C temperature difference. Therefore, running the CPU at 100 MHz and 1MHz generates about 1.8°C temperature difference. The temperature change is much smaller than that can be generated in a general purpose CPU because the Nios II processor is a soft core processor and hence has larger area and lower power density.

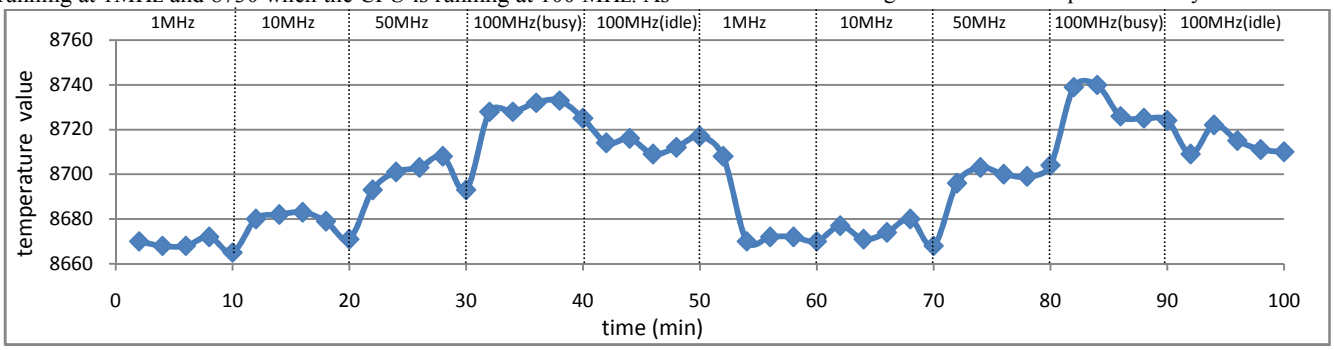


Figure 11 .Processor's temperature change under different working frequency

6. Conclusion

In this paper, we proposed an FPGA-based distributed computing platform using Altera Nios II soft-core processors. The system consists of multiple FPGAs with each FPGA configured as a multi-core system. Each core has its own temperature sensor and has the dynamic frequency scaling capability. The platform is designed to support research in dynamic power/thermal management. Our experiment verifies the correct functionality of the distributed computing platform and evaluated the communication latency between cores and between FPGAs. It shows almost linear performance improvements as the number of cores increases.

7. References

[1] T.Dorta, J.Jimenez, J.L.Martin, U.Bidarte, and A.Astarloa, "Overview of FPGA-Based Multiprocessor Systems", in *Reconfig'09*, pp.273-278,Dec.2009

[2] D.Atienza, P.G.D Valle, G.Paci, F.Poletti, L.Benini, G.De Micheli, and J.M.Mendias, "A Fast HW/SW FPGA-Based Thermal Emulation Framework for Multi-Processor System-on-Chip,", *DAC'06,* pp.618-623, Jul.2006.

[3] N.Fujii and N.Koike, "Work in Progress-Developmemt of a New Multi-CPU Parallel and Distributed Processing Experiment Platform for Remote hardware laboratory," in *Frontiers in Education Conference, 2009. FIE '09. 39th IEEE* ,pp.1-2, Oct.2009

[4] M.Manzke and R.Brennan, "Extending FPGA based Teaching Boards into the area of Distributed Memory Multiprocessors," in *Proc. WCAE '04*, Article No.5, 2004

[5] L.Yan and etc., "Performance Evaluation of the Memory Hierarchy Design on CMP Prototype Using FPGA,"in *ASICON'09*, pp.813-816, Oct.2009

[6] A.Tumeo,C.Pilato,G.Palermo,F.Ferrandi and D.Sciuto, "HW/SW Methodologies for Synchronization in FPGA Multiprocessors," in *Proc.FPGA'09* , pp.265-268, 2009

[7] J.J.Martinez-Alvarez and etc., "A Multi-FPGA Distributed Embedded System for the Emulation of Multi-Layer CNNs in Read Time Video Applications," in *CNAA'2010*, pp.1-5, Feb.2010

[8] A.Kulmala, O.Lehtoranta, T.D.Hamalainen and M.Hannikainen, "Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC," in *RASIP Journal on Embedded Systems,* vol.2006, issue 1 , Jan.2006

[9] A.Tumeo and etc., "Prototyping Pipelined Applications on Heterogeneous FPGA Multiprocessor Virtual Platform," in *ASP-DAC'2009,*pp.317-322, Jan.2009

[10] L.Shang, R.P.Dick and N.K.Jha, "SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol.26,pp.508-526, Mar.2007

[11] A.Bhattacharjee, G.Contreras and M.Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," in *Proc.ISLPED'08*, pp.335-340,2008

[12] Micrium, "uC/OS-II Kernel," available at: http://www.micrium.com/page/products/rtos/os-ii .

[13] Altera Corporation , "Mailbox Core," Quartus II Handbook Version 9.1 Volume 5, Nov.2009

[14] Altera Corporation , "Mutex Core," Quartus II Handbook Version 9.1 Volume 5, Nov.2009

[15] Altera Corporation , "Clock Multiplexing," Quartus II Handbook Version 10.1 Volume 1, Dec.2010

[16] Altera Corporation, Phase-Locked Loop Reconfiguration (ALTPLL_RECONFIG) Megafunction User Guide, Aug.2010

[17] Altera Corporation, Phase-Locked Loop(ALTPLL) Megafunction User Guide, Nov.2009

[18] Altera Corporation, Clock Control Block (ALTCLKCTRL) Megafunction User Guide, Sep.2010

[19] Altera Corporation , "Avalon Memory-Mapped Bridges," Quartus II 10.0 Handbook, Volume 4 , Dec.2010

[20] P.Chen, M.C.Shie, Z.Y.Zheng, Z.F.Zheng, C.Y.Chu, "A Fully Digital Time-Domain Smart Temperature Sensor Realized With 140 FPGA Logic Elements," in *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol.54,pp.2661-2668, Dec.2007

[21] L.Benini, A.Bogliolo and G.De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans on VLSI*, vol.8, issue3,pp.299-316,2000

[22] Y.Ge, P.Malani and Qinru Qiu, "Distributed task migration for thermal management in many-core systems," *DAC2010*, pp.579-584, Jun.2010

[23] Altera Corporation , SOPC Builder User Guide , Dec.2010