MIPS-FPGA

Chethan Kumar
chethank001@ntu.edu.sg
G1502140K

Nihar
nihar001@ntu.edu.sg
G1502131J

Ajith
jose0029@ntu.edu.sg
G1502145F

November 10, 2015

Abstract

Usage of On-chip networks to communicate between processors is a great way to improve performance in a Multiprocessor system. The aim of our project is to implement a multiprocessor system which uses a Network on Chip (NoC) for establishing communication between processors (nodes) with the help of send and receive instructions. In our project we use MIPSfpga soft processor as it allows implementation of User Defined Instructions (UDI) through its corExtend interface. Our design supports instantiations of upto 256(16 X 16) cores. Smaller multiprocessor systems with number of cores upto 8 result in a sustained rate 2x that of the larger systems (above 8 cores) because of less complexity of the network. Execution rate of a 32 core system is found to increase by 2 as the FIFO depth increases from 64 to 512. The worst case latency of a 128 core system is 110 cycles which is thrice that of a 32 core system at 0.5 injection rate. Also, we notice that a design with UDI is taking an extra M9k block and 10 more logic registers when compared to the original design.

1 Introduction

Multiprocessor systems provide high performance as they parallelize the applications. Memory organization in multiprocessor systems can be categorized into two groups. Symmetric shared-memory multiprocessor (SMP) and Distributed shared-memory multiprocessor (DSM). The former uses a centralized shared memory, along with a bus interconnect to establish communication between multiple processors contained in the system. Symmetry here indicates that each processor sees the same memory organization. In this case the hardware must ensure propagation of updated values through the system which is quite complex to manage. The latter uses an interconnection network to transfer data between processors in the form of messages/packets. The interconnection network which allows message passing is called NoC. In this case the shared memory is divided into smaller slices of memory and each one of them is made local to that particular processor. DSM essentially indicates that the address space is shared. For the communication to be established through the NoC, send and receive instructions are required. MIPSfpga soft processor from Imagination technologies allows implementation of any new functionality via it's UDI interface. We implement send and receive instructions and connect to the existing NoC module and test our design for various dimensions and traffic patterns to analyze the latency and throughput characteristics. We implement the design on an Altera Cyclone IV FPGA on the Terasic DE2-115 board. We estimate the FPGA resource utilization of the design and also perform power analysis of the same.

2 Motivation

SMP systems suffer from problems like Cache coherence and resource contention which lead to a bandwidth bottleneck. There are several advantages of going for a NoC over Shared memory architecture. Firstly, the cache coherence problem will be eliminated with the help of directory based protocol since handling snooping based protocol for the hardware becomes very hard as the number of nodes in the system increases. Secondly, all the links inside the NoC can operate simultaneously on different data packets there by providing higher level of parallelism and as the complexity of the system increases NoC provides enhanced scalability. Furthermore, the hardware is comparatively simpler which leads to reduction in the gate count thereby reducing the static power consumption as it is proportional to the area . Dynamic power consumption also will be less as the design is less complex than the SMP systems .

3 Design

3.1 User Defined Instruction

The special2 opcode of the MIPS Instruction Set Architecture allows us to implement up to 16 UDIs. Two new instructions namely SEND and RECEIVE are added to the existing MIPS instruction set. Figures 1 and 2 below depict the Send and Receive instruction formats respectively.

OPCODE[31:26]	SRC[25:20]	DEST[19:6]	FUNC[5:0]
011100			011000

Figure 1: Send Instruction Format

OPCODE[31:26]	SRC[25:20]	DEST[19:6]	FUNC[5:0]
011100			010101

Figure 2: Receive Instruction Format

3.1.1 Send

Basic functionality of the send instruction is to move data from one MIPSfpga processor to other through NoC. Figure 3 shown below is the data flow model for send instruction. UDI module at the sender side will wait for instruction valid signal from instruction decoder and then reads data from

the communication RAM's location whose address is specified as a part of the instruction, generates NOC packets and hands it over to the NoC network interface only if the network is free otherwise it buffers packets into the FIFO instead of stalling execution of the next instruction to avoid processor overhead. Packets stored in FIFO are transferred when network becomes free.

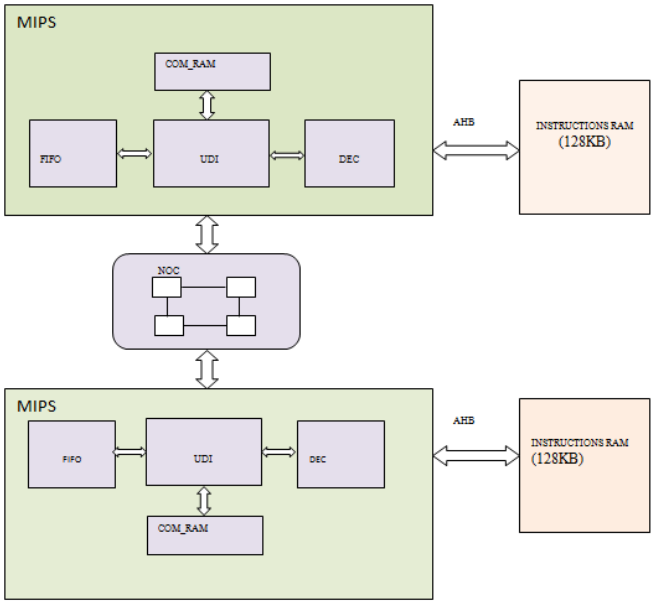


Figure 3: Send instruction data flow

3.1.2 Receive

Receive instructions are executed to update the register file of the receiver MIPSfpga processor with the data received from the other MIPSfpga cores. UDI at the receiver side will check for the valid packets from NOC and buffer them in another internal scratch pad RAM. Once NoC finishes sending data, the receive UDI is enabled. The data is then read from the RAM and written to the register file.

4 Experiments

4.1 Testing

We used ModelSim-Altera Starter Edition 10.3c to simulate our design. There are two phases involved in testing. In the first phase of testing, instruction RAM configuration files loaded with machine codes for send/receive instructions are generated for individual MIPSfpga cores. In the second

phase, modelsim project is opened through tcl script and simulation is initiated.

./mips.tcl MODE REPEATITION PATTERN RATE

MODE This parameter specifies the number of MIPSfpga cores to be instantiated. There are seven modes, from MODE0 to MODE6 each corresponding to 4, 8, 16, 32, 64, 128 and 256 MIPSfpga processor cores respectively.

REPEATITION This parameter is used to generate packets in large number. If specified as 1, for 4 MIPSfpga cores it generates 4 x 32 packets where 32 is the number of scratchpad register locations which is fixed. 2x number of packets can be generated by making this parameter 2.

PATTERN This argument is used to specify the type of network traffic pattern to be used in UDI packet generation. Our design is capable of generating three different types of traffic patterns like RANDOM, LOCAL and TORNADO by passing values of 1, 2 and 3 respectively to this parameter.

RATE This argument is used to select the injection rates between 0.5 and 1.

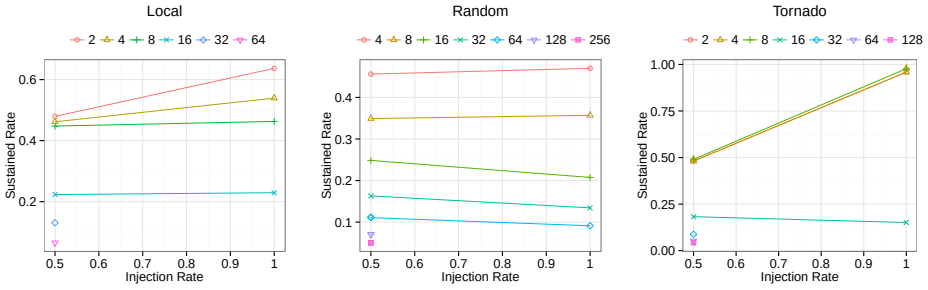
Code generation Addition of new instructions makes it impossible for the compiler to generate a machine code with new user defined instructions. We wrote a script to facilitate the purpose of machine code generation. It generates individual instruction RAM configuration file for each MIPSfpga core. This script takes few parameters from top level script as mentioned below.

- mips - Number of MIPSfpga processors to be used in simulation
- pat - Traffic pattern

Simulation During simulation each MIPSfpga core executes initialization machine code to initialize its resources i.e, Data cache, Instruction cache and TLB before executing send/receive instructions. At the end of simulation various log files are generated such as mips_bw.csv, fifo_depth.csv, data_error.txt and fifo_overflow.txt. These log files are used to analyze the throughput, data correctness and FIFO depth sensitivity of the system.

4.2 Throughput Measurement

Throughput test is performed for 2, 4, 8, 16, 32, 64, 128 and 256 core system at two different injection rates 0.5 and 1. Comparison between injection rate and sustained rate for local, random and tornado traffic patterns are as shown in Figure 4. From the plots it can be noticed that as the system complexity increases sustained rate of the system decays. Local patterns have relatively higher sustained rate as compared to other two patterns because, each core sends packet to an other core which is located within the specified window size where as in case of tornado and random, the destination address is completely random. For 2,4 and 8 core system tornado pattern results in sustained rate which is almost equal to the injection rate because in these systems each MIPSfpga core will have a fixed destination.



(a) Local (b) Random (c) Tornado

Figure 4: Throughput

4.3 Sensitivity test

FIFO depth sensitivity test is performed to find out minimum fifo depth required for the given system to avoid stalling of processor. This test is executed for 4, 8, 16 and 32 core systems at 0.5 injection rate. Our evaluation results clearly show that FIFO depth is directly proportional to the instruction execution rate. For a system with 4 cores, fifo with 32 locations is more than sufficient to avoid stalling where as a 32 core system requires a fifo of depth 512. In case of 32 core system a fifo of depth 32 results in fetching instruction once in 5 clock cycles instead of two clock cycles. This test proved that to have a better execution rate for a larger system we need larger fifo.

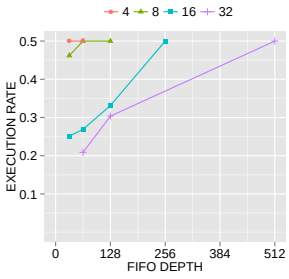


Figure 5: FIFO depth sensitivity plot.

4.4 Latency

Figure 6 depicts the density plot of packet latency for 32, 64 and 128 core system at 0.5 injection rate. This plot helps us in calculating worst case and an average packet delay of the given system. When the complexity of the system increases the worst case latency also increases which is evident from the density plot. For a 32 core system most of the packets take less than 15 cycles to reach the destination core where as in case of 128 core system it is 25 cycles. In the worst case scenario, the 128 core system's worst case latency is 110 cycles which is thrice that of the 32 core system.

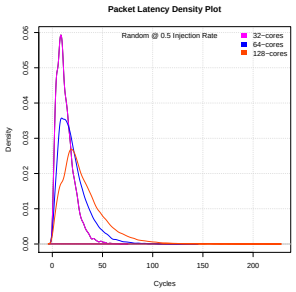


Figure 6: Latency of a Packet.

4.5 Area Utilization

Table 1: Resource Utilization

Resources	Original	Modified
Total Logic Elements	15,326 / 114,480 (13%)	15,208 / 114,480 (13%)
Total Registers	7,425 / 117,053 (6%)	7,435 / 117,053 (6%)
M9k blocks	403 / 432 (93%)	404 / 432 (94%)
Embedded multipliers	0 / 532 (0%)	0 / 532 (0%)
Total PLLs	1 / 4 (25%)	1 / 4 (25%)
Total memory bits	3,223,936 / 3,981,312 (81%)	3,224,000 / 3,981,312 (81%)
Total pins	57 / 529 (11%)	57 / 529 (11%)

Table 1 indicates the area utilization of a single core MIPSfpga system and a comparison between the original and modified MIPSfpga processors. We used Altera Quartus 14.1 tool to find out the resources utilized in the FPGA. After adding the UDI instructions, there has been a very less rise in the number of registers utilized because the addition of extra logic and memories for send and receive instructions accounted for increase in 1 M9k block.

4.6 Power measurements

Table 2 indicates the power consumption information of a single core MIPSfpga system and a comparison between the original and modified MIPSfpga processors. We used PowerPlay Power analyzer tool of Altera Quartus 14.1 to analyze the power consumption. The overall power dissipation in the

Table 2: Approximate Power Dissipation

Power consumption	Original	Modified
Total Thermal Power Dissipation	511.65mW	502.27mW
Core Dynamic Thermal Power Dissipation	351.56mW	342.23mW
Core Static Thermal Power Dissipation	104.58mW	104.53mW
I/O Thermal Power Dissipation	55.51mW	55.51mW

original case is a little higher than the modified one since all unused UDI signals were driven always high.

5 Performance Improvements

MIPSfpga comes with a 2 way set associative cache which when enabled helps in improving injection rate of packets from 0.1 (with cache disabled) to 0.5. MIPSfpga lacks instruction pre-fetching mechanism which is a drawback in terms of throughput. Whenever the processor encounters a cache miss the controller fetches 4 instructions from memory and during this time, the execution unit will go to idle state which leads to 50% reduction in injection rate. To counter this issue we padded previously executed instructions in the 4 cycle gap in order to achieve an injection rate exactly equal to 1.

6 Conclusion

In this project we show how to implement send and receive instructions to exchange data in a DSM multiprocessor system. We prove that the smaller multiprocessor systems result in a better sustained rate and the execution rate is directly proportional to the FIFO depth. We show that the worst case latency increases with the complexity of the system. Our design consumes a little more area and less power as compared to the original design.

References

N. Kapre and J. Gray. Hoplite: Building austere overlay nocs for fpgas. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–8, Sept 2015. 10.1109/FPL.2015.7293956.

Imagination technologies. Mipsfpga getting started guide. URL `http://community.imgtec.com/download-notes/mipsfpga-getting-started-material-version-1-0/`.