DEVELOPMENT OF A NETWORK ON CHIP FOR PARALLEL PROCESSING SYSTEMS

Kazuto Tanaka, Satoshi Iwanami, Tuneki Ohashi, Takeshi Yamakawa and Chikara Fukunaga
Mathematics and Information Sciences
Tokyo Metropolitan University
Hachioji, Tokyo, Japan
contact email: tanaka-kazuto1@ed.tmu.ac.jp

ABSTRACT
We have developed a network (called TPNET) which is adaptable for any parallel processing systems. It consists of several core processors and a router. A process element in a parallel processing system is a processor called TPCORE2, which has been developed by the authors’ group. Since this core processor can execute full set of the transputer instruction set, we can describe a software system using the parallel processing language occam. Occam is based on theoretically a model called Communicating Sequential Processes (CSP). If a parallel system can be described in occam language, and work fine, it will be regarded as free from any deadlocks or livelocks which will be intrinsically hidden in a parallel system. We can construct simply a secure parallel processing system in this way.

Each processor can be connected to a router, and we can achieve a dynamic configuration of the network topology by controlling the router. The basic communication protocol in TPNET is IEEE 1355. An assured and efficient network can be constructed despite the structural simplicity of the protocol.

With characteristics discussed above and with an efficient interrupt processing system in TPCORE2, we propose this TPNET as a basic framework for high performance embedded systems used widely in various industrial fields.

KEY WORDS
Parallel Processing, Parallel Programming, Routing, Interconnection Network

1 Introduction; TPNET

We have developed an IP core framework of a compact network system of which the installation target will be an FPGA (Field Programmable Gate Array). The network system (we call TPNET hereafter) consists of several processors and a router. The network connections can be flexibly changed according to the purpose to utilize TPNET at the FPGA implementation. Since it is intended to be expanded all in all in an FPGA, an NoC (Network on Chip)[1] is realized eventually.

The network architecture we have adopted in the NoC is a mixture of so called “direct networks” type in which each node has its own routing mechanism and point-to-point links to other nodes and “indirect networks” in which each node is connected to a router and the router has point-to-point links to other nodes.

The “direct networks” have been universally used for building large scale systems. However, consumption of the area and energy for physical connection in a chip increase significantly, and make serious limitation for the system development. Furthermore inefficiency such that messages from a node passes usually several intermediate nodes before reaching its destination makes a software driven in a node complicated. In order to overcome these problems and to reduce the chip resource as much as possible while maintaining the high network performance, we have designed the main network with the “indirect network” scheme by introducing a router at the center of the system, and deployed several sub-networks with direct processor connection on top of the main network.

In fig. 1, we show an example network structure with several processors. In the figure the processors are denoted as TPCORE2 which is a processor developed by the authors’ group. Its characteristics is discussed in the next section.

![TPNET Example](image)

Figure 1. An Example of a network configuration in TPNET. Core processors are named as TPCORE2

A node of TPNET can be a single processor or a sub-network of a bunch of the processors. TPCORE2 has three identical interfaces (called as link). A simple sub-network can be built by connecting one of the link interfaces to one of other TPCORE2. A sub-network topology with several processors can be also flexibly organized in this way.

For designing the router, we have chosen the IEEE 1355[2] as the principal protocol in the network. Conse-
sequently TPCORE2 has an another link interface that makes message passing with this protocol. The design concepts of our router have come from the SpaceWire router[3] although actual technical implementation has been different from one presented here. The protocol SpaceWire is extended from the IEEE 1355 and standardized to be used in a payload of a space craft or a space station.

The router we have constructed has a crossbar with the wormhole packet switching. Although the virtual channel system is normally installed in a wormhole router itself for flow controlling[4], we have installed it in TPCORE2. We intend to use this virtual channel for not only the router flow control but also flexible mapping of software structure onto the actual network topological structure. Both the virtual channel system and the flow control mechanism implemented in the IEEE 1355 make the TPNET insusceptible to deadlock.

We intend this TPNET chip to be used in a realtime embedded system for multi-interrupt environments with many sensors and processors which will be realized in, for example, the electronics engine control system in an automobile or in a space vehicle and various (attitude and/or steering) control systems in a robot. Parallel processing mechanism and event driven inherent in TPNET and in TPCORE2 will be appropriate for a system for multi-interrupt systems.

2 Parallel Processor; TPCORE2

The processor TPCORE2 has been constructed in 2009. This processor is an update of our previous processor called TPCORE which has been made in 2004[5]. Our primary motivation to make TPCORE is to develop a processor which executes the instruction set of transputer T425 of Inmos Ltd.[6]. Consequently a parallel processing system described in occam programming language[7] can be executed in a single or network of several TPCOREs compatibly. TPCORE works with basic clock frequency of 50MHz when the core is implemented in an FPGA of XILINX Virtex4. It has four identical communication link interfaces. A network with several TPCOREs can be constructed using these links. The data transfer rate is designed as 50Mbit/s (the number equals to the clock frequency).

The occam language is a programming language developed also by Inmos Ltd. devoted to describe a parallel system on transputers. The theoretical model of parallel processing in the language is based on Communicating Sequential Processes (CSP) invented by C.A.R.Hoare in 1978[8]. In this theory two parallel processes exchange their common parameters through channels. A channel can be a memory object if two processes are in a single processor and a link if two processes are distributed in different processors but they are handled uniquely in an occam program.

The top level structure of TPCORE is shown in fig. 2. It is segmented into four main partitions. The CPU and the link block can independently access to the memory through the memory controller. The link block is an interface block for four links to be used for external connection with other TPCOREs. The protocol used in this link (called OS-link) is a simple one in which 11 bits are used for one byte transfer (eight bit data plus two bit for start and one for stop mark). The DMA (Direct Memory Access) transfer of messages between two TPCOREs via a link can be achieved owing to this structure. We could make smooth development from TPCORE to TPCORE2 also because of this simple structure. For the TPCORE2 development, we have concentrated to modify the link block and some minor modification of the memory control although we have kept the another blocks as ones installed in TPCORE.

We have designed the CPU structure almost from scratch. We have designed our own stack mechanism, registers, micro-code ROM, mechanisms of process management, interrupt handling and communication control with our own way. If we could keep the core body as small as possible, we can implement so many TPCOREs to install in an FPGA. We could actually implement total 16 TPCOREs in a XILINX Virtex4 chip. A topologically complicated network can be realized if the number of processors in the FPGA increases. By confining complicated descriptions such as one to control internal states of CPU and their transitions in micro-code ROM, we could reduce significantly consumption of resources in the FPGA by installing the micro-code ROM in the internal memory space while keeping the logical block as small as possible.

In the following subsections we list the features newly installed in TPCORE2.

---

\[1\] We have installed 16 TPCOREs actually in type XC4VLX160 of Virtex4 series.
2.1 IEEE 1355 for the Link Protocol

We have implemented a newly a link whose communication protocol is IEEE 1355 in TPCORE2 with reducing the number of OS-links to three from four. In TPCORE2 while we keep the OS-link, which will be used for the subnet network configuration, and we have added the IEEE 1355 for the global network connection link.

2.2 Event Driven Interrupt Handling

TPCORE2 has a simple interrupt mechanism. The interrupt signal is regarded as a channel input from the another process from the software point of view. An idea for the interrupt handling is, therefore, based on the channel communication in the CSP theory. The interrupt detection process can be processed in parallel with other processes. The latency of the interrupt (time from the interrupt signal reception to start of actual interrupt processing) is 22 clocks, i.e., 440ns. TPCORE2 has the capability to accept maximum 13 different hardware interrupt inputs (channels) simultaneously. Two levels of the priority can be set for these inputs.

2.3 Virtual Channel

We have also added so called a virtual channel mechanism[9] newly in TPCORE2. The virtual channel is a system to assign several software links (virtual channels) on one physical link. Every packet of the IEEE 1355 protocol must have the destination address in the header part. If we can assign an own address to each process running in TPCORE2 of the another side, the data packet can eventually reach to the destination process successfully. In this case an overall software process works as if one (virtual) processor occupies one independent (virtual) link. In TPCORE2 we have installed a hardware virtual channel processor to manage this mechanism.

3 Router

While topological configuration of a subnet using conventional links can not be modified dynamically, we can modify overall network structure dynamically thanks to introduction of the router. The number of ports is currently set to four. Either a single TPCORE2 or TPCORE2 subnet can be a node processor. A representative processor should be connected to a port at the router when the node consists of the subnet. The crossbar in the router uses a wormhole packet switching. The crossbar switching from the source to the destination is done within two clocks after it receives the packet. The priority of the switching is a priori set for ports if two packets simultaneously are transferred to an identical port. In order to choose a correct port from the given destination address of a virtual channel, we install a mapping table in the router for storing the mapping information between the port number and an interval of virtual channel numbers for each port. The table can be accessed from a Host PC with a serial port connection. In order to simplify the routing algorithm, we assign number of virtual channels within an interval to a physical port. The table should be configured before a software system is executed on TPNET.

The block diagram of the router hardware is shown in fig.3. It is operated in 50MHz clock. Every port is fully compliant with the DS-link (physical layer of IEEE 1355 protocol). The hardware block for a port consists of DS-Transmitter (Xmitter)/Receiver (Rcvr), Token Analyzer and Packet Analyzer blocks. The Token Analyzer checks the validity of a data token (10 bit for one byte data) in a packet with its parity received. It makes also flow control in a case that a FIFO buffer to save tokens in a packet are short in size. The Packet Analyzer block is used for a input packet to extract the destination address stored in the header. The address extracted is passed to the crossbar control block, and the actual port is determined from the address from the mapping table installed in this block.

The duration required for a packet transfer in the router is observed 34 clock (680ns) as it is shown in fig. 4. In this diagram the first and second signal lines show signals observed at the input and output ports, respectively.

Figure 3. Block diagram of TPNET Router with four ports

Figure 4. Signals observed in the router. Signals observed input and output ports are shown in the first two wave lines
4 Performance

For performance measurement, we have focussed on effects of multiplicity of the virtual channel in packet transfer and the router switching characteristic.

In measurement of data transfer we have setup a simple network with two TPCORE2s and one router. In order to make situation simple, we have fixed one TPCORE2 as data sender, and the another one as receiver. Programs running in both TPCORE2s have the same number of parallel processes and this number is identical with the number of the virtual channels introduced in the link. Each parallel process in the sender TPCORE2 sends independently packets to the corresponding process in the receiver one. A packet transfer measurement is proceeded as:

1. data (fixed length) for a packet is prepared in a source process,
2. number of the virtual channels is fixed, this number is equal to the number of processes in each TPCORE2, namely one virtual channel is connected for each pair of the sender process and the receiver process,
3. parallel processes start simultaneously at the sender side, a timer is started at this moment, the timer used is one embedded in TPCORE2 (100ns precision),
4. each process repeats the data transfer using own virtual channel in given number of times,
5. data transfer means the sender sends a packet, the receiver receives it and sends back an acknowledgement, and the sender receives it,
6. during the repetition the transfer direction, the packet length and the destination process for a sender process are kept fixed, and
7. the timer is stopped at the end of the last process (end of the processes are not simultaneous), in the sender TPCORE2.

We have measured this time duration with various number of channels (1, 2, 3, 4 and 8) and packet length (1, 8, 16 and 32 bytes). In the IEEE 1355 protocol the maximum number of bytes in a packet is limited to 32. If one must send a data more than 32 byte, one must divide data in multiple packets. The measurement of packet transfer up to 32 byte is, therefore, enough to get the performance for this TPNET. The throughput is calculated as "packet length in byte" × "number of repetitions" × "number of virtual channels" / "time measured".

In fig. 5, we show the result of the measurements. We express the throughput (Mbit/s) versus the number of the virtual channels concurrently introduced in this figure. Four data plots come from the measurements for four different packet sizes. Lines connected between any two adjacent data points are just for a guide to eyes. In our current system, always the virtual channel is used for the data transfer even if the number of multiplicity is only one. The maximum number of the virtual channels is currently set to eight. In all four graphs we find that the throughput becomes saturated as the number of the virtual channels becomes greater than three. We can not expect higher efficiency data transfer even if number of the virtual channels is increased. At a given number of virtual channels, however, the throughput increases linearly as the number of bytes in a packet increases as we can see from the graph.

Figure 5. Packet Transfer rate Mbit/s[Mbps] versus number of virtual channels

Figure 6. Processing time in the router for packets with various size

In fig. 6 we show the wave form of relevant signals actually observed in the router. Four diagrams shows commonly timing difference of a packet signal at the router input port and it at the output port. As the byte length in-
creases, the router processing time is also increased. Since one bit is processed in one clock in the router and the bit length for n byte transfer in the IEEE protocol is given as $n \times 10 + 10 + 4$ in which additional 10 bit and four bit are the header and the end of packet, and the router overhead for the switching is 34 clock as discussed in the previous section, the whole time needed for one n byte packet transfer is estimated as $n \times 10 + 48$. The measured time is, however, actually longer than the estimation with $n \times 5$ clock regularly. In the router we have eight stage FIFO (First In First Out) buffer in Token Analyzer block for each port. One byte data is stored in one stage of the FIFO. If this FIFO is empty, the router issues a signal to request the data transfer towards a packet sender, which is called the flow control. One flow control takes 40 clock in the current scheme. Number of the flow controls occurred in an n byte packet transfer is estimated to be $n/8$. Thus other $(n/8) \times 40 = n \times 5$ clock is just spent for this flow control. As the time consumption in the router has been clarified reasonably from this discussion, we conclude that the router worked consistently in the packet transfer process as have designed.

5 Summary

We have developed an NoC system called TPNET, which will be a basic framework for a parallel processing environment. We intend the system to be implemented in an FPGA. The performance measurement has been done with a simplified TPNET which consists of only two TPCORE2s and one router, and the highest throughput 18.9 Mbps was observed in packet transfer of 32 byte length with the number of the virtual channels greater than three. Although we can not make simple comparison of the network performance with other routers published, it has been reported in [10] that a SpaceWire router with eight links achieved the transfer rate of 130M bps. If we use other processors than TPCORE2 with higher clock frequency, we may achieve the bit rate close to this value. In this case we must change drastically the method for parallel processing in a processor. We stick to, however, use TPCORE2 as the main processor in the network in order to write down a parallel processing system program using the occam language without introducing any operating system, to make a sub-network quickly using embedded OS-link interfaces, and to use the virtual channel which will bring the flexibility of software mapping onto the hardware network topology and insusceptibility to deadlocks in parallel system into an NoC system. In a multi-interrupt system, a processor which has a capability to handle an interrupt in event driven way is indispensable. TPCORE2 has facility to process interrupts in this way, and can run the interrupt service process with the any other processes in parallel. We expect the application field of TPNET will be extended gradually. Actually we discussed in [11] the possibility to use of our network system with TPCORE in an aerospace environment.

An NoC with many TPCORE2 will be, therefore, an appropriate choice for a realtime embedded system for multi-interrupt environments.

6 Acknowledgements

We wish to express our gratitude to Mr. Kazuto Matsui for his advices and suggestions throughout this work.

References


