Haupt-Seminar Report

Reliable Networks-On-Chip in the Many-Core Era

Institut für Technische Informatik, Universität Stuttgart

NoC monitoring and error detection

Supervised by Dipl.-Inf. Adan Kohler

Submitted by Hou, Jie

June 09, 2009
Contents

1 What’s NoC and why do we need monitoring for NoC? ......................................................... 3
  1.1 What’s NoC? .................................................................................................................. 3
  1.2 Why do we need monitoring for NoC? ................................................................. 3

2 A generic event-based NoC run-time monitoring system .............................................. 4
  2.1 Event Model ............................................................................................................... 5
  2.2 Monitoring Probes..................................................................................................... 8
    2.2.1 Generic probe Architecture ................................................................................... 8
    2.2.2 Æthereal Probe Architecture ............................................................................... 9
    2.2.3 The probe’s model ............................................................................................. 10
  2.3 Monitoring traffic management ................................................................................. 10

3 Transaction Monitoring in NoCs .................................................................................. 10

4 The impact of NoC monitoring ..................................................................................... 14

5 Brief introductions about Error Detection .................................................................. 16
  5.1 Hamming code and CRC code .................................................................................. 17
    5.1.1 Hamming Code ..................................................................................................... 17
    5.1.2 Error detection with CRC (Cyclic Redundancy Check) ...................................... 18
  5.2 Error Detection Methods ......................................................................................... 19

6 Summary ....................................................................................................................... 21

7 Reference ...................................................................................................................... 21
1 What’s NoC and why do we need monitoring for NoC?

1.1 What’s NoC?
As everyone knows, developments in semiconductor technology have made very complex large scale System-on-Chip (SoCs) design available. Usually, in a System-on-Chip, interconnection between different Intellectual Property (IP) cores is achieved by means of shared bus architectures. However, nowadays each new SoC generation integrates more processing elements, more features and more new functionalities. With this increasing complexity, the system maybe needs extremely high capability in computation and communication. So, it is obvious that we can’t continue to use shared bus architectures, which will affect performance of the overall system. Also, they can’t present a scalable solution to existing problems in the communication [1].

In order to solve these problems, on-chip point-to-point distributed interconnection networks or Networks on a Chip (NoCs) have been proposed. Unlike the shared bus architectures, the key communication method in NoC architecture is to implement interconnections of different IP cores using on chip packet-switched networks [1].

NoCs are composed of three components: Routers (switches), Links and Network Interfaces (NI). Routers are the switching elements that are responsible for forwarding data packets from one router to another one. Links are the connection parts between different routers, and they are usually bidirectional. Network Interfaces are the wrapper between the router and processing element (PE). They have two different communication operations. Firstly, NIs can collect the data from all kinds of elements that are attached to them, packetize, add the header and tail information and send the processed packet into the attached router. Secondly, they receive the packets from the attached router [1].

1.2 Why do we need monitoring for NoC?
NoCs have been researched for several years, and they become more and more popular and important in recent times. People from the world have proposed several concrete examples of NoCs, such as Æthereal, Xpipes, QNoC and Mango. Normally, NoCs require sophisticated tools to aid in design time. Furthermore, with increasing complexity there is a strong need for runtime monitoring for NoC. The monitoring must be accounted for in the design phase, and it also affects the design flow of the original NoC [2] [3]. In the section 4, we will see these.

Observability is an important issue in the system debugging. As we know, SoCs that are based on NoC have two main functional parts: computation and communication.
Usually there are also two kinds of observabilities: computation observability and communication observability. Without proper computation and communication observability, the task of debugging can not be finished. The basic requirement of debugging for a system is to have a monitoring system. Traditionally, when people implement debug tools and methods for a system, they usually only consider the computation part. And the area of computation observability has been widely reached. For example, ARM’s embedded trace macrocell (ETM) can provide real-time information about core internals, which is able to trace instructions and data at core speed nonintrusively. However, now many SoCs contain many processors, memory elements and IPs, different elements maybe want to cooperate with each other, and they need to communicate. Obviously, communication also plays an important role in the NoCs. However, usually the on-chip communication observability has been ignored. Moreover, with the development of NoCs, the on-chip communication becomes more sophisticated, e.g. some NoCs can perform dynamic adaptive routing algorithm. And during the operation of NoC, some problems also can arise, e.g. congestion or deadlock. We also know that: in the traditional shared bus architecture, it usually has some central arbitration component for bus utilization and arbitration. However, in NoCs there is no such component. Moreover, multiple parallel communication paths may work simultaneously in NoC, which also increase complexities about debugging. In a word, we need communication observability and monitoring service for NoC [3].

2 A generic event-based NoC run-time monitoring system

The authors of [3] propose an event based run-time monitoring system for NoC-based SoCs. This solution is a generic NoC monitoring service that can be implemented for any NoC. This monitoring system needs hardwared probes, and these probes can capture runtime information and functional data and generate event based information. Usually, the NoC monitoring services are as follows: how to configure hardwared probes that are attached to network components, i.e. routers or network interfaces, and how to manage monitoring traffic. We can see architecture of a NoC monitoring system in the figure 1. In this figure, R is router, NI is network interface and P is probe. Here probes are attached to routers. All the monitored data is transported to Monitor.
2.1 Event Model

As above mentioned, this monitoring system is event-based. What is an event? The authors of [3] define that an event is a happening of interest, which occurs instantaneously at a certain time. For NoCs, all the monitored information is modeled in the form of events. We can imagine that: during the operation of NoC, it can generate a lot of events. We’d better classify them into several types. Within the NoC, we can have either user traffic or NoC internal traffic. Furthermore, these traffics can be either control traffic or regular data traffic. If we use these information as axes, we get the following figure.

In the figure 2, each quadrant represents one type of events. However, the fifth type can not be seen in this figure. It is called Monitoring Service Internal Events. Normally, for all NoCs, people use these five types to describe events. But, each type for different NoCs may have different subtypes. In the later part, we use Æthereal NoC as example to describe these. Firstly, we’d better look at Æthereal NoC briefly. The Æthereal is just a protocol proposed by Philips, and it is used to realize the correct functionality of the NoC. According to [4], Æthereal NoC is summarized as follows: it runs at 500 MHz and offers raw bandwidth of 2GB/s per link in a 0.13 micron technology. For Æthereal NoC, the designers can choose the topology arbitrarily. Æthereal NoC offers two types of transport layer communication services to IPs, and they are best-effort (BE) and
guaranteed throughput (GT) services. GT services have higher priority than BE services, and they use slots that have been reserved. GT services also can make sure that the data can be transported to the destination in time and successfully, e.g. all kinds of important configuration data can be transported using GT mechanism. Æthereal NoC uses a TDMA mechanism to allow several virtual connections to share one physical link. For Æthereal NoC, an event is just a tuple: Event = (identifier, timestamp, producer, attributes). We can use the identifier to describe the type of an event. The timestamp defines the time at which the event is produced. The producer is to describe which element on the chip generates that event. And attributes contain the useful payload of events. Usually different events have different attributes, and an event type also can have several different attributes. In the following part, we can see these. Like event’s description, each attribute can be formed in this way: Attribute = (attribute identifier, value). Now let’s look at these five types for both general NoCs and Æthereal NoC.

**User Configuration Events.** When different IPs attached to NoC want to communicate with each other, there must be a connection existing between them. This connection may be useful for debugging or other purposes, so we need to record it. Usually, we just record this communication in system level (or user level), e.g. the time or both parties of this connection, which is called User Configuration Events. For Æthereal, there are two different types of events that belong to this type. The first one is called connection opened event, which means a certain connection between different IPs has been opened. In this situation the attributes are connection identifiers, connection types (e.g. narrow-cast, broadcast) and the ports between which the connection has been established. The other one is called connection closed event. Obviously, this means that the connection is tore down because of some reasons. The attributes here are just connection identifiers.

**User Data Events.** When we want to check the specific interactions of the components in the NoCs, we can use additional hardwares (e.g. hardwared-probe that will be introduced in detailed way later) to sniff or spy some functional data from the routers or from the links. We call this way as User Data Events. Normally, it can sniff the information about flits, packets and even a whole message. For Æthereal, Sniff event usually means that getting flit information from the queue of a router. This type of event contains two similar kinds. The first one is GT Sniff Event, which means we can use some sniffers that can detect GT flits. The attributes of this type are GT flit itself and the identifier of the queue where the information was sniffed. The BE Sniff Event is similar to GT Sniff Event except that it only can detect BE flits.

**NoC Configuration Events.** We have known that User Configuration Events can provide system level information about communications. However, sometimes we need more
detailed information in NoC level for system debugging, e.g. which slot in the slot table of a router can be used. We call the monitored information that are related to control and NoC level as NoC Configuration Events. For Æthereal, this type of event refers to Reserve Slot Event, Free Slot Event and System Packet Arrived Event. A Reserve Slot Event shows that a certain slot in the slot table of a router or NI has been reserved for some communication. The attributes of this event are the reserved slot number and its value. The value means that: in which port the data goes into the router and from which port the data leave the router. And, a Free Slot Event means that a slot table in a router or NI has been released when the communication was finished. The attribute of this event is only the slot number. System Packet Arrived Event means that programmed packets being used for the configuration of network and the attributes of this event are the whole packets. Usually, we can use the attributes of this event to see what exactly the system will do.

**NoC Alert Events.** During the operation of NoCs, there may be some problems such as: buffer overflow, congestion, deadlock, livelock and so on. Sometimes these problems can give us serious problem. For example, in hard-time real time system, the timeliness is extremely important. So, it is imperative for us to detect these problems and give the system some alert according to different problems. This kind of events is called NoC Alert Events. For Æthereal, this type of event also contains many different kinds. For simplicity, two of them are introduced. Firstly, Queue Filling Event shows that the number of queues a router has and the states of these queues (e.g. these queues full or not). For example, a router has four ports, and each port has a queue. We have four attributes for it, in which we usually store how much the queues have been used. Then, let’s see an End-to-end Credit 0 Event. It means that: When we talk about the flow control about NoCs, there is one called Credit-Base flow control. We use credit to show how many flit buffers are still available. If the credit is 0, this means that some IP connected to this router must be blocked. At this time, this type of event is generated. The attribute of this event is the connection identifier.

**Monitoring Service Internal Events.** In this situation, we call all the events that are used by monitoring services for its own purposes (e.g. ordering of these events) as Monitoring Service Internal Events. For Æthereal, a probe attached to NoC can generate many kinds of events. These events are normally in the timestamped way, so we can get the ordering of these events easily. This ordering is also important for system debugging. Here we also need to understand a concept about synchronization. As we know, the probe of the NoC may be used for a quite long time, during this time it will generate a large number of events. So the orders of these events may be quite large. It is expensive to use many bits to store the event number. So the number of bits is
typically limited and thus the event number will eventually wrap around. At this time, synchronization happens. This synchronization is important from the system’s point of view. In a system, commonly there is only a physical clock. If we want to know every event’s physical time, we must use synchronization. For example, the limit of a timestamp counter is 4. The sequences of first four events are (0, 1, 2, 3); then the counter wraps, if later another four events are generated, their sequences are also (0, 1, 2, 3). From the system’s view, it can not distinguish the events in the corresponding positions in first sequence and in the second one without the help of synchronization. This type of events doesn’t have attributes.

2.2 Monitoring Probes

2.2.1 Generic probe Architecture

The authors of [3] also propose hardwired probe architectures for general NoCs and Æthereal NoC. Now let’s look at what is a generic hardwired probe and how it works. In the Figure 3, we can see an architecture that is composed of three parts: Sniffer (S), Monitoring Network Interface (MNI) and Event Generator (EG). We can see that: the Sniffer can get some information from Router (R) and send it to EG, then EG can generate some type of event and send it to the router through MNI, and finally the generated event is sent to monitoring service access (MSA) point. A MSA is an IP, and it has two functionalities. One is to control the configuration of those monitors (e.g. probes) attached to the elements of NoC at run time, and the other is to receive all the monitored data from them. In this figure, we can see that the router can be used to transfer either programming data for probes or monitored data from MNI.
2.2.2 Æthereal Probe Architecture

Now we can see a specific probe-architecture of Æthereal. It also contains three parts: S, MNI and EG. Firstly I describe this picture, and then I will explain the function of each part in detail. Usually, a probe can be attached to a Router or an NI. Here it is attached to the Router. The Router can send the programming data through the input port (I1), output port (O5) to MNI. Then the MNI can depacketize the received programming data and configure itself by the port CNIP (Configuration Network Interface Port). And it also can configure the EG via the mmio (memory mapped I/O) port that is connect to the EG’s Configuration port. From the MNI’s point of view, this mmio is a memory-mapped master port. The Sniffer can send the probed information from Router to EG. The EG then generates the event and transfers them to MNI, which can packetize the events and then transfers them to Router’s input port (I5). Finally, the Router can send these packetized events to MSA from the port o2.

Sniffer (S): The function of sniffer is to get info from router and send it to the EG. It can be attached to either the router or the links. Here it is attached to the router.

Event Generator (EG): The function of EG is to generate the timestamped events according to received information from sniffer. Previously we have already known that the event is described in this way: Event = (identifier, timestamp, producer, attributes). For Æthereal, we use 8 bits to describe identifier, 16 bits to describe timestamp and 8 bits to describe producer. Usually, the sizes of the attributes are different. So, the sizes of different events are also different. In the above section, we have known some types of events and their attributes. For today’s NoCs, it’s enough to use 8 bits to describe identifiers of events. Producer specifies from which router the events were generated. In the figure 4, we know the EG contains three ports: Event Port, Data Port and Configuration Port. Event Port is a memory-mapped master, and it usually has a queue that can store some events temporarily.
**Monitoring Network Interface (MNI):** The function of MNI is to depacketize the programming data from router, packetize the event data from EG and play an important role in the communication between Router and EG. Because of the design of MNI, the EG can be looked as an IP.

### 2.2.3 The probe’s model

In the previous section we have known the architecture of the hardwired-probe, now let’s see how to use it. Firstly, we need to determine when the probes are configured. There are three different types: at the time of initial configuration of NoC, at the time of reconfiguration of NoC or at run-time. For Æthereal, because the probes have a memory-mapped slave port, we can configure them at run-time. Except that, the connection between MNI and MSA must be set up like normal standard Æthereal connections, because the packages that contain generated events must be sent to MSA. Moreover, the probes also need to know what types of events they should monitor, how many attributes of each event should contain and how to enable or disable probes.

### 2.3 Monitoring traffic management

There are two kinds of traffic: probe configuration traffic and event traffic. Probe configuration traffic means that: the programming traffic required to configure the probe and the traffic that is used by system to build a connection for MNI and NI of the router that requests the monitoring service. Event traffic is the data that is generated by the EG in the probe. Usually, there are several ways to transfer the event traffic to the router that requests monitoring service, e.g. building a dedicated NoC or using the existing NoC. If we use the existing NoC, the generated event traffic may interrupt the user data traffic of the existing NoC. However, if we build a dedicated NoC, the cost of elements and area is high. In section 4, we will see how monitoring system affects the design flow of an original NoC. For Æthereal, we assume that it can use the existing NoC to transfer the traffics. As we know, Æthereal has two kinds of communication services: GT and BE. If all the traffics are generated by probes using GT connections, in this way even if the network is congested, they still can reach the MSA. This assures the run-time behavior of monitoring service. However, if the generated traffics use BE connections, when the network is congested, they are not guaranteed [3].

### 3 Transaction Monitoring in NoCs

In the second part we have seen a generic monitoring-system’s architecture, if we replace the Event Generator using a Transaction Monitor (TM), then we can monitoring the system in transaction level. What is a transaction? The authors of [4] define that: a
NoC monitoring and error detection

transaction is a data structure containing information about transfer, e.g. a structure with memory address, transfer size and payload. For NoCs, the interconnected IPs use transactions to communicate with each other. Transactions are performed on connections. Usually one transaction comprises one or more messages. These messages can be either read message or write message. For example, a request from an IP can be a read message, and the response can be some write messages that contain the data for this request. Why do we need transaction monitoring for NoC? If we use above event based monitoring system, we can get a large number of abstract information. Because all these monitored data need to be transported to MSA, this means: more data need more time and cause more burdens for system’s bandwidth. In order to improve the speed of system level debugging, we’d better raise these data from abstraction level to transaction level.

According to [5], researchers propose a Transaction Monitor (TM) to solve that problem for Æthereal. Before we see TM, firstly we’d better know something about Æthereal packet format (Figure 5) and flit format (Figure 6). As we know, the Network Interfaces chop message into pieces of maximum length and add a header and a tail to each of these pieces. Each of these pieces is called a packet. Usually, packets are of different lengths. A packet can comprise several flits, and each flit corresponds to one TDMA slot in Æthereal. A flit contains three 32-bit words, and each of these words has two bit sideband information (see in figure 5). The first 2-bit sideband information shows whether the flit is BE or GT and whether it contains a packet header or not. The second 2-bit shows how many payloads this flit has. And the final 2-bit shows whether it is the end of packet or not.

Figure 5. Æthereal packet format [5]  Figure 6. Æthereal flit format [5]

Now let’s look at the TM architecture in a detail way.
From the figure 7, we can know that: the sniffer can get all the information about links that a router has and transfer them to the TM. The link selection block can be used to choose one of those links for future use. TM has four transaction monitor processing blocks. One or more of those blocks can be chose at run-time via the enable/configuration block. The TM uses two ports (MP, SP) to communicate with MNI. MP means a master port, which is memory-mapped master that can send transaction monitor data to MNI. SP means a slave port, which is memory-mapped slave. The MNI can send programming data to configure TM via SP. The TM has four different modes according to four different transaction levels. Each of those four transaction monitor processing blocks corresponds to one mode. Now let’s see these four modes in a little detail way [5].

**Raw Mode.** From the figure 7, we can know that after link selection block only one link can be chosen. For Æthereal, we know that many connections can use one link by means of TDMA. So many flits of different connections can be sent either directly to MNI or to GT/BE filtering block. If all the flits are directly sent to MNI, there may be some potential problem because of limited physical bandwidth. We also know that Æthereal’s communication services contain two types: GT and BE. So we can use GT/BE filtering block to distinguish whether the flits are of GT type or of BE type, which can alleviate that potential problem. This filtering is possible, from the Æthereal package format, we know that the first 2-bit sideband information can indicate whether this flit is GT or BE. In a word, one can observe the raw traffic as it can be observed over a specific link, and after this process one can get all the traffic of one service type over a specific link.

**Connection-based Mode.** This mode can be used to provide full observability on all bits of a selected connection. After the GT/BE filtering block we know that you can either
get all the GT flits or BE flits on one link. We also know that, for Æthereal one link means several different connections. Connection-based Mode just wants to transport flits about one of those connections. From Æthereal package and flit formats, we know that: in the package header, there are some information about the path to destination NI/Router and the queue identifier of the destination NI/Router. The combination of all above described information can determine a virtual connection. The connection filtering block can be configured to choose the connection that we want to examine. For example, in the centralized programming Æthereal NoC, there is usually a configuration master that is connected to a router. We also know that all NIs have their identifiers. The configuration master can send many programmed data to several NIs, if we only want to monitor the configuration of one NI that we are interested, we can use this NI’s id to configure the connection filtering block in TM. Then only the flits belong to that connection can be filtered. This mode is useful in case that all details of a certain connection want to be examined. And this mode is also efficient for alleviating the potential problem about system’s physical limited bandwidth.

Transaction-based Mode. After connection filtering block, we can get all the flits of a certain connection. The purpose of transaction-based mode is just to identify different messages belongs to one connection. How to identify the messages? Usually it is difficult to detect the start of a message, because they are packed in packets without any alignment. For Æthereal, it uses hardware modules for message depacketization. Those hardware modules assume that the first packet over a connection carries the first message header that is always after the package header. And the first message header contains the size of this message. So now the problem is easy. If the message start is detected, the rest of message can be got using the size information in message header easily. Once a message is finished, then another message header is detected. I want to say another important thing about this mode is: from above assumption we must be sure that the transaction monitor can get the first packet over a certain connection that needs to be monitored. Otherwise this mode will not work any more. This problem can be solved by always setting and enabling the transaction monitors before the actual connection is set. For Æthereal, this method is possible. Because it can use a MMIO method to configure the TM at runtime, and it always configures the TM before user connection configuration. Moreover, we can add some additional options into the depacketization block, e.g. we can distinguish the read messages and write messages.

Transaction Event-based Mode. After the depacketization block, we can identify all the messages over a certain connection. However, not all of these messages are useful. For “unimportant” message, after the abstraction block, only a transaction event generates. A transaction event just contains command, address and number of words in a message.
For example, a write message with 7 words of payload has a total of 9 words for the entire message. If this message is not important, we can abstract it in an event of 2 words (command, address, number of this message’s payload) and discard the contents of the message. The traffic in this mode is lower than in other modes.

4 The impact of NoC monitoring

In the section 2 and 3, we have seen a generic monitoring system and a specialized one. When we add monitoring service to an existing NoC, it must have effect on the existing NoC. And monitoring elements also need to be considered in the design flow. In this part, we will have a look at two extreme architectural constructions about adding monitoring service to an existing NoC. The first one is to use a separate interconnect for the monitoring data, and the second one is to share all the existing NoC resources for both user data and monitoring data. Now let’s analyze these effects from the design flow point of view.

Firstly we’d better have a look at the typical NoC design flow briefly. According to [2], we know that the typical NoC design flow is normally split into four steps (see in Figure 8(a)): topology selection, mapping, path selection and slot allocation. Each step is dependent on the previous step. Before these four steps, some important prerequisites are as follows: (1) when you want to design a NoC, you must have a general idea about the whole system’s communication requirements (2) how many sets of IPs to be connected to the NoC. Normally, in the topology selection step, the router network together with the bordering NIs is generated according to the previous communication requirements. In the mapping step, the binding of different IPs to the bordering NIs is achieved based on the generated topology and the IP specification. In the path selection step, paths are allocated for all the specified communication flows. Finally, in the slot allocation step, each of the communication flows gets its own slots based on the TDMA method.

Then let’s look at the first extreme method that is to use a separate physical interconnect for monitoring data. An example is in Figure 9(b), and its design flow for monitoring NoC is in Figure 8(b). From figure 9(b), we can know that for each router in user NoC there is an attached probe. We call this form a fully probed NoC. The functions of the monitoring NoC are to transport the monitored data from probes to Monitoring Service Access point (MSA) and send the configuration data from the MSA to the probes attached to routers in user NoC. We also can know that each probe and the MSA are attached to the monitoring NoC by the NIs. The NoC design flow is applied twice in this method. One is for user NoC and the other is for monitoring NoC. The one
for user NoC is the same as 8(a). For monitoring NoC, the design flow can be seen in figure 8(b). Here we need to consider monitoring communication requirements firstly. However the monitoring communication requirements are dependent on the user NoC, because the number of debug IPs (e.g. hardware-probes attach to routers) is related to the number of routers or NIs in user NoC. Fortunately, the user NoC can be fixed before monitoring NoC. We also can know that, the monitoring NoC should have similar topology as user NoC, because each probe directly connects to a router in user NoC and indirectly connects to a router in monitoring NoC by a NI. And we also know that mapping can be determined by probe placement in the user NoC. Therefore, only path selection and slot allocation have nothing related to user NoC design and need to be done again. In this method, we can see that monitoring NoC and user NoC will not be interrupted by each other. However, the drawback of this method is that its area cost is high [2].

(a) Original NoC        (b) Separate NoCs        (c) Shared NoC

Figure.8 Design Flow Modifications [2]

(a) Original NoC        (b) Separate NoCs        (c) Shared NoC

Figure.9 Monitoring Transport Options [2]
The second extreme method is to share the same physical interconnect and all physical resources for both user traffic and monitoring traffic, but we also try our best to keep the NoC user traffic and the monitoring traffic separated. For user NoC, after user communication requirements are computed and mapping is done, which mean that a framework for user NoC is achieved. After that, we need to compute the monitoring communication requirements and decide how many debug IPs (e.g. hardware-probes) are needed. From figure 9(c) we see that probes can be connected to the user NoC by means of the existing user NoC’s NIs. The mapping of the probes to existing NIs is based on the closest NI principle. Then path selection and slot allocation are done based on all communication requirements of both user data and monitoring data. Figure 8(c) show this design flow. There are two possibilities using this method. The first one is that everything fits on the existing user NoC well. In this situation, it is obvious that we have the lowest area cost. The second possibility is that: the combined monitoring communication requirements can not fit on the existing user NoC, it is clear that a new user NoC need to be generated again. We can extend the topology of previous user NoC, and which results in the need of more routers. So we can know that the number of probes also increases. Finally, we must compute the monitoring communication requirements and IPs. If they also can not fit the new generated topology, all previous steps need to be done repeatedly. In the worst case, maybe we can not find a suitable solution. We can see these changes in the design flow in the figure 8(c). There is also another problem, user traffic and monitoring traffic may be disturbed by each other [2].

5 Brief introductions about Error Detection

In the sections 2 and 3, we have known that a monitoring system can provide communication observability about system, but it has no idea whether the transmitted data over the monitored communication path is getting corrupted or not. Moreover, as we know, the development of technology in VLSI makes the system’s supply voltage and the dimension of system decrease obviously. And the wires become more unreliable because they are affected by the noise sources such as crosstalking, coupling noise and process variation, which can result in glitches, delay effects, logic errors. So it is necessary for us to think about some methods to detect or correct such errors in the low level. One of the mostly widely used techniques is to add some redundant information to the original data. As an example, error detecting and error correcting code can be used to protect system. In the digital communication area, people use some kinds of codes (e.g. parity code, Hamming code, CRC code and so on). For NoCs’ interconnect communication, these codes are also applicable. I read several papers
briefly, and I find that people usually use Hamming code and CRC code frequently. So now I want to introduce these two kinds of codes.

5.1 Hamming code and CRC code

5.1.1 Hamming Code

**Definition 1**: Let $x$ and $y$ be strings of the same length $n$ over an alphabet. The *Hamming distance* between $x$ and $y$ is defined to be the number of places in which the $x$ and $y$ differ. It is denoted by $d(x,y)$ [6].

**Definition 2**: Let $C$ be a code with at least two codewords. The *minimum Hamming distance* $d(C)$ of $C$ is the minimum of the Hamming distance between all possible pairs of codewords of that code [6].

**Properties of minimum Hamming distance**:

1. To correct $n$ single-bit errors, a code $C$ must have a minimum Hamming distance $d(C)$ of at least $2n + 1$, $d(C) \geq 2n + 1$.
2. To detect $n$ single-bit errors, a code must have a minimum Hamming distance of at least $n+1$, $d(C) \geq n+1$

**Definition 3**: In mathematical terms, Hamming code is a binary linear code. For each integer $m > 2$ there is a code with parameters: $[2^m-1, 2^m-m-1, 3]$. $(2^m - 1)$ is the total length of Hamming code. $(2^m-m-1)$ is the length of information bits, and Hamming code always has the minimum distance 3.

So we can know that Hamming code always can detect 2 errors and correct 1 error. Now let’s look at a specific example: $(7, 4)$-Hamming code. This code is 7 bits long, and it has four data bits (D) and three parity bits (P). And the code can be described in vector as the codeword $C$.

$$C = [D7 \ D6 \ D5 \ P4 \ D3 \ P2 \ P1]$$

These parity bits ($P1, P2, P4$) and data bits ($D7, D6, D5, D3$) should always have some relationship as follows:

$$S1 = D7 \oplus D6 \oplus D5 \oplus P4 = D7 + D6 + D5 + P4 = 0$$
$$S2 = D7 \oplus D6 \oplus D3 \oplus P2 = D7 + D6 + D3 + P2 = 0$$
$$S3 = D7 \oplus D5 \oplus D3 \oplus P1 = D7 + D5 + D3 + P1 = 0$$

They are also can be described in the form of matrix product: $S = MC^T$.

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$M$ is called parity-check matrix.
We know the 7-bit binary code can generate 128 codewords, but not all 128 codewords satisfy the above relationships, only 16 codewords (e.g. 1100110) are valid. Now let's see an example about how it can correct one error bit.

For example, suppose the above code 1100110 is sent and a single bit error occurs such that 1110110 is received, (1110110) = (1100110) + (0010000) = C + E. E is an error vector.

Transmitted message 1100110  Received message 1110110
S1 = 1 + 1 + 0 + 0 = 0  S1 = 1 + 1 + 1 + 0 = 1
S2 = 1 + 1 + 1 + 1 = 0  S2 = 1 + 1 + 1 + 1 = 0
S3 = 1 + 0 + 1 + 0 = 0  S3 = 1 + 1 + 1 + 0 = 1

101 points directly to the bad bit, because 101 in binary equals to 5 in decimal.

We can see that:  S = M(C + E)^T = ME^T + MC^T = ME^T

\[
\begin{pmatrix}
1 \\
0 \\
1
\end{pmatrix}
= \begin{pmatrix}
1111000 \\
1100110 \\
1010101
\end{pmatrix} \times \begin{pmatrix}
0010000
\end{pmatrix}^T
\]

Now let's look at an example about how to detect 2-bit errors. In this situation, we just assume: E = (0 0 1 0 1 0 0).

\[
\begin{pmatrix}
1111000 \\
1100110 \\
1010101
\end{pmatrix} \times \begin{pmatrix}
0100100
\end{pmatrix}^T = \begin{pmatrix}
1+0 \\
0+1 \\
1+1
\end{pmatrix} = \begin{pmatrix}
1 \\
1 \\
0
\end{pmatrix}
\]

The result is non-zero vector, this means that some error has occurred. The error is not correctable, because some other errors maybe also can lead to the same result as here. We also know that the sum of any two columns of matrix M can not generate a zero vector, which means it can guarantee that 2-bit error can be detected [7].

5.1.2 Error detection with CRC (Cyclic Redundancy Check)

The CRC also add redundancy information to message bits in order to detect errors, which can be seen as follows.

<table>
<thead>
<tr>
<th>message bits</th>
<th>CRC bits</th>
</tr>
</thead>
</table>

The exact number of CRC bits and its makeup depend on the generating polynomial. Because CRC is widely used in many areas, and there are also many kinds of generating polynomials, e.g. \(x^8 + x^7 + x^3 + x^2 + 1\) is a standard generating polynomial used by CCITT. Next, I'll give you an example about how CRC can detect errors.

We assume a message represented by polynomial \(G(x)\), and \(P(x)\) is used for generating polynomial. In our example, \(G(x) = x^5 + x^4 + x\) (binary 110010), \(P(x) = x^3 + x^2 + 1\) (binary 1101). Then we can calculate the transmitted message using \(G(x)\) and \(P(x)\).
Step 1: Multiply the message $G(x)$ by $x^3$, where 3 is the number of bits in $P(x)$, $[x^3G(x)] = x^3(x^5 + x^4 + x) = x^8 + x^7 + x^4$.

Step 2: Divide the $[x^3G(x)]$ by $P(x)$, the remainder $C(x) = x^2$ (binary 100).

Step 3: Yield the message polynomial $F(x) = [x^3G(x)] + C(x) = x^8 + x^7 + x^4 + x^2$ (binary 110010100).

This generated message polynomial $F(x)$ is used to be transmitted. In the receiving side, once the whole message $R(x)$ is received, which can be check easily by $R(x)/P(x)$. If the remainder is 0, this means that the transmitted message is not corrupted, $R(x) = F(x)$.

However, if a single bit error occurs in bit position K in the transmitted message $F(x)$, then $R(x) = F(x) + E(x)$, where $E(x) = x^K$. In this situation: $R(x)/P(x) = [F(x) + E(x)] / P(x) = F(x)/P(x) + E(x)/P(x) = 0 + E(x)/P(x) ≠ 0$. So the one-bit error is definitely detectable. We also know that any adjacent-double-bit error also can be detectable [7].

### 5.2 Error Detection Methods

In the section 5.1, we have seen what are Hamming code and CRC and how they can detect errors. Now let’s look at some schemes for error detection and correction for NoCs. In a simple retransmission scheme, the sender adds error detection codes (e.g. CRC, parity code) as redundant information for the original information data, once the receiver receives the transmitted message. It can check whether the received data is correct or not. If there is some error, the receiver can ask the sender to transmit the message again. In another scheme (for simplicity there is at most an error can occur), if the sender adds the error correction codes (Hamming code) to the original data, once the receiver detect the received data is not correct, then the receiver can correct the received data by itself. This scheme is really efficient, but it is very limited. Another scheme is to combine the retransmission and correction. The authors of [8] propose three classes of error recovery schemes: end-to-end, switch-to-switch and hybrid error detection methods. They assume that the NoC has the architecture as follows: each core has one sender and receiver network interface, the router has some input queue and use credit-based control flow. Now let’s look at these methods in a detail way.

**End-to-End error detection**

In the end-to-end scheme (in figure 10 we can see its architecture), we add the encoder for each sender NI and decoder for each receiver NI. The encoder uses some error-detection codes (parity, Hamming code and CRC) to add redundant information to the original data, then the decoder can detect whether the received data is correct or not. From the figure 10, we also can know that sender NI usually has one or more packet buffers to store those packets that has been transmitted. In a request-response
transaction, the receiver can send the ack or nack signal back to sender according the received data. If the received data isn’t correct, the receiver will drop the received message and the sender needs to retransmit the information. Usually there is a timeout mechanism scheme for sender’s retransmission. End-to-end control flow is also called network layer control flow [8].

![Switch-to-switch retransmission architecture](image1.png)

**Figure.10 End-to-end retransmission architecture [8]**

**Switch-to-switch error detection**

We can see the architecture for switch-to-switch (link level) error detection schemes in figure 11. From the figure, we can know that: sender NI has an encoder, each switch contains a decoder and the receiver doesn’t have encoder or decoder. The encoder uses some error-detection codes to encode useful information. Normally, the switch has two sets of buffers: queuing buffer and credit-control-flow buffer. But here, in order to support this scheme, we add a retransmission buffer to store the data that maybe need to be retransmitted, which support the switch-to-switch transmission mechanisms. We also use some other redundancy model (e.g. triple modular redundancy) for control signals such as ack line. There are two types of switch-to-switch schemes, one is at flit level and the other one is at packet level. The capacity of retransmission buffer is different for these two schemes. At flit level, the retransmission buffer requires the capacity of \((2N_L+1)\) flits for full throughput operation, where \(N_L\) is the number of cycles required to cross the link between two adjacent switches. The reason is that in credit-based control flow, it takes one cycle to generate the credit signal, a flit needs \(N_L\) cycles to transmit from the one switch to another switch, and the credit signal also needs \(N_L\) cycles. However, at packet level, the retransmission buffer requires the capacity of \((2N_L+f)\) flits for full throughput operation, where \(f\) is the number of flits that a packet has [8].

![Switch-to-switch retransmission architecture](image2.png)

**Figure.11 Switch-to-switch retransmission architecture [8]**
Hybrid error detection

In this scheme, if there is only single-bit error in the flit, the receiver can correct it. If there are multiple-bit errors the receiver will ask the sender to transmit the information again using end-to-end retransmission mechanism [8].

6 Summary

In this seminar report, firstly I introduce a generic monitoring system based on event model and its application on the Æthereal NoC. This kind of monitoring system uses monitoring probes, and the architecture and programming model of these probes are also introduced. Then I introduce a specialized monitoring system in transaction level. This monitoring system uses transaction monitors (TM) instead of event generators (EG). We also see four working modes of TM. Then I also analyze the impact of monitoring system on the existing user NoC. Finally, I introduce two kinds of codes (Hamming code and CRC) and their properties in error correction and detection. Three error detection methods are also introduced briefly.

Through writing seminar report, I also learn some other knowledge, such as how to use google to search useful science and technology papers, how to summarize the useful information and so on. I think all these knowledge are useful for me.

7 Reference


