RELIABLE NETWORK-ON-CHIP ARCHITECTURE FOR RELIABLE APPLICATIONS

Michael Grieco (260969688)

I. INTRODUCTION

As the number of communicating elements in a system grows, Network-on-Chip (NoC) architectures are a scalable solution for multiprocessor systems-on-chip (MPSoC). One advantage of using a NoC is the ability to layer higher-level services on top of the provided architecture, similar to layers in a computer network [1]. This can create an avoidable overhead if services are constant throughout most of the communication protocols.

For example, including redundancy in a system is necessary for system reliability. Implementing triple-module-redundancy (TMR) by having multiple copies of a module is a common method of ensuring accuracy. One downside of including redundancy is redundant communications, or the need to replicate data so that all module copies receive it. When using TMR, the number of commands a host must send increases by a factor of three. Synchronizing that is a challenge, especially when using complex interconnects. A command host should only need to send a single command, being completely blind to the redundancy mechanism. This emanates high-level layers in a network protocol stack being blind to the lower-level ones, which the NoC can achieve. An additional obstacle in TMR is de-centralizing the module copies to ensure that spatially localized errors do not cause faults in multiple copies, thus propagating an error to the voter. Using a NoC allows the three copies to be spatially-independent network clients.

This project researched and designed an implementation for integrating redundancy into a NoC architecture at the link control level. This leverages the flexibility of NoCs while improving their robustness to make them a more feasible option for reliable systems. When testing with a small data streaming application on a mesh NoC, the custom hardware model achieves 3x speedup over the typical architecture with a small increase in memory requirements. The SystemC code is available under an MIT license.

II. RELATED WORK

While the concept of a NoC is not novel, researching their reliability is a newer subject. Li et al. [2] introduced several architectures to implement fault-tolerant NoCs. Specifically, they experimented with routing algorithms, topology reconfiguration, and routers containing adaptable data paths. Each address a general need of hardware fault-tolerance and how to design NoCs to achieve those goals. However, the strategies do not address application-level redundancy. The implementation in this paper can work in tandem with mechanisms introduced in [2] to yield a more robust system. Weichslgartner et al. [3] implemented a framework to specify runtime constraints on a program's resource usage. They designed an invasive-NoC (*i*-NoC) using a hypervisor to manage clients connected to the NoC. This allows for constraints such as reliability, which the hypervisor models at runtime. This was a general approach that can optimize various applications. However, for deterministic protocols fixed at design time, using a dynamic management system creates unneeded overhead. The *i*-NoC demonstrates that NoCs can be used in safety critical systems. This project will take these concepts while ensuring more deterministic performance by implementing reliability control in hardware.

Fault-tolerant mechanisms like TMR, however, has lots of literature addressing the most effective ways to improve system reliability. This project will apply techniques introduced in [4]. Caplan et al. analyzed different signal compression techniques to compare signatures of data streams. They determined that cyclic redundancy check (CRC) was the most efficient hardware-implemented fingerprinting technique. Meyer et al. [5] determined efficient implementations of redundant operations through distributed temporal redundancy. Their system would not execute a third redundant thread unless the first two did not agree. The paper also covered dynamic fingerprinting, or how to compare unsynchronized fingerprints. This project uses these concepts to design the voter module embedded in the NoC architecture.

III. APPROACH

This project aimed to customize the architecture of NoC components to integrate application-level redundancy support in hardware. The idea is to have the client initiating redundant communications connected to a router via a NoC adapter issue a single packet to the adapter. The adapter can then decide to make redundant copies of the packet to send to pre-determined redundant copies of the recipient module. The adapter is a more suitable location for this control logic as opposed to routers because this feature can be optional based on the connected client. Keeping routers as uniform as possible simplifies network configuration and avoids putting bottlenecks on specific routers.

Specifically, this means that each adapter will read control bits from the link control packet to indicate whether to send redundant copies. At design time, the designer must map the control bits to the copies of the module through a local list of tile addresses in an adapter. The adapter stitches the three global tile addresses to the relative addresses received from its tile. This generates three separate NoC packets, containing identical data and link control except for the destination address. With typical NoC mesh architecture, a router would buffer each of these packets in virtual channel FIFOs, each originating from the corresponding tile. This means that the router could only send one of the redundant packets per clock cycle, not improving much upon the idea of sending three separate packets from the tile. To address this, this project introduces the concept of read-only, write-only, and read-write ports on a NoC router.

A. *NoC Port Types*

This project defines read-only (RO) port is a port that a router can read from but not write to; a write-only (WO) port as one that a router can write to but not read from; and a read-write

(RW) port is a port with duplex traffic. Using this, the crossbar within the router will map all readable ports to all writeable ports. Fig. 1a shows a typical NoC mesh router, with one tile port and four cardinal direction ports, all RW ports. In the proposed design, the adapter copies packets from the tile while only employing a single port to write data back to the tile. Combining this with the definitions, the adapter has three RO ports and one WO port. To simplify this, the adapter can interface with the router using two RO ports and one RW port.

In this design, as Fig. 1b displays, the router crossbar must consider seven ports to read from: the four cardinal directions, and three tile ports. Each port can communicate with the five output ports. Introducing the extra ports allows the redundant ports on the adapter can each issue their own packets, as shown in Fig. 2a. Hence, all redundant packets can issue to the router in the same clock cycle, solving the issue above.

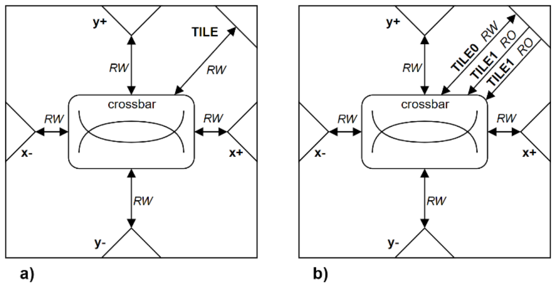

Fig. 1. NoC router architecture, with direction names boldened and port access types italicized. (a) Typical NoC mesh router, (b) The NoC mesh router with additional ports for redundancy support. The router in (b) functions like it has 7 readable ports, but the three TILE*X* ports are hardwired to the same output of the adapter, except for the destination address in the link control packet.

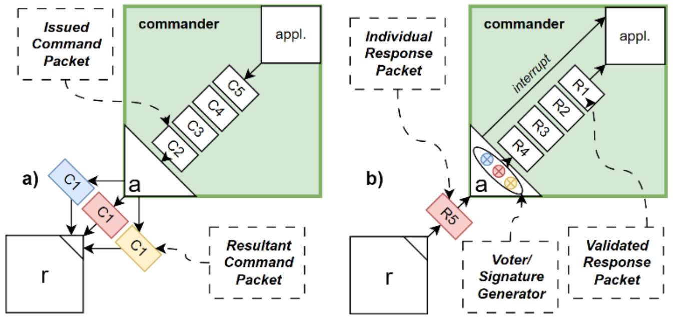

Fig. 2. Customized NoC adapter, with the adapter shown as “a” and the router shown as “r” for that NoC tile. (a) Issuing command packets once to the adapter who makes redundant copies. (b) Receiving response packets, where the adapter internally generates signatures, votes, and can raise interrupts for invalid votes.

B. *Data Reception*

Given that module copies are distinct clients in the network, their synchronization is not guaranteed. For instance, the experimental setup (Fig. 4) evaluates a NoC with the module copies being different distances (i.e., number of router hops) from the commander. Hence, there is an inherent latency difference in the module copies, meaning that the output data is not synchronized. As a result, performing voting on incoming packets is not applicable, as those packets may not correspond to the same output. To solve this, a system could buffer the entire output from each module copy then perform voting, an expensive approach. Alternatively, systems can perform fingerprinting to calculate signatures of input streams [4].

This project used the fingerprinting approach to customize the reception port on the tile adapters for the commander as summarized in Fig. 3. In general, input streams must receive enough packets to complete a checkpoint, at which point the voter compares CRCs for that checkpoint.

When a packet arrives at the adapter, the adapter determines if the source was one of the redundant modules. If not, the adapter forwards the packet to the application layer. Otherwise, the adapter uses the packet to update the current CRC, increments the counter for the source module, and adds the packet to the corresponding buffer for the current checkpoint. The counter indicates if the input stream passed a checkpoint. If so, the adapter saves the CRC in a separate buffer indexed by the checkpoint number. Additionally, if the module was not the first one to arrive at the checkpoint, voting can occur. The adapter compares the updated CRC to stored CRCs from the other input streams for that checkpoint. If there is a majority, the adapter forwards the completed checkpoint buffer for the current input stream to the application as validated data. An additional point of interest is that the majority does not require all three input streams to have reached the checkpoint. If the first two CRCs for a checkpoint agree, there is a majority, and the adapter forwards the validated data buffer. Hence, the third input stream reaching the checkpoint does not trigger voting or data forwarding.

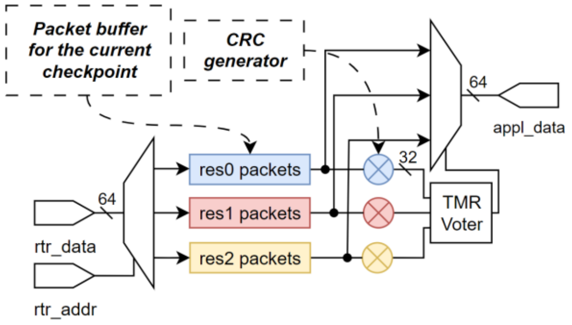

Fig. 3. Adapter input control logic. The address of the incoming packet determines the corresponding packet buffer and CRC update. The voter stores previous CRCs and will select a buffer to forward to the application packet-by-packet once two CRCs agree for the incoming checkpoint.

For each redundant module, the adapter must hold a buffer large enough for a single checkpoint. Since the current input stream forwards to the application, the input streams must only stay up to date with the corresponding module. The adapter need not store any history of the input streams, the CRCs compress and preserve this. However, the consequence is that there must be a separate CRC counter for each checkpoint the application can go through. There is a tradeoff of number of checkpoints and the size of a checkpoint. With the smallest possible checkpoint, the data buffer would be a single packet, but the adapter would need to calculate and persist CRCs after each packet.

IV. Experimental Setup

This project implemented a SystemC model of a NoC system, as shown in Fig. 4. There are two versions of the system, corresponding to the two versions of the NoC adapter. The first will be a typical adapter, and the second will be customized with built-in redundancy support using the proposed design. Both systems will satisfy TMR by implementing three copies of a worker module as different clients of the NoC. For the benchmark model, the command host will perform redundant communications and then validation at the application level, acting as an optional service on top of the NoC. For the customized system, the NoC will integrate the redundancy check into the adapters. Hence, the command host need only issue one command to its adapter, and it will assume the received data has already passed through a validation system. Each router has enough memory for 32 buffer entries per readable and writeable direction, storing the data packet and link control wires. The redundant voter has 4 checkpoints, with a checkpoint size of 4 packets. The data packet size is 64 bits.

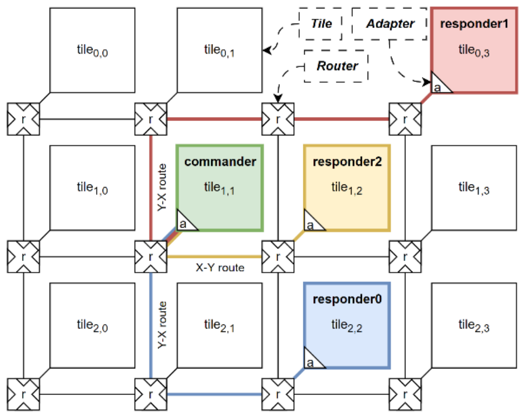

Fig. 4. NoC system under test. All blank tiles have no activity, hence the SystemC model does not consider network congestion.

The evaluation of both systems included performance and reliability metrics. Fig. 5 presents a summary of the simulation environment around each tile in the modeled NoC. The SystemC simulation time provided a latency of the whole application. For more specific timing statistics, the model implements a global packet store. Whenever a tile issues a packet to the adapter, the store adds the packet with the timestamp and an associated class. Once the target application receives and validates the packet, the application notifies the packet store of the reception. The packet store then deletes the packet from the table and logs the duration from issuance to acceptance.

To measure fault tolerance, the simulation injects faults into the system. Using strategies in [6], the SystemC model registers variables and signals that can experience single event upsets (SEU). Every specified interval, the simulation halts, and the fault-injection system will randomly flip bits. Before simulation starts, modules register pointers with the global fault injector, allowing that data to be upset. In the registration, each variable has a probability of upset, defined as probability of a potential fault per byte per timestep. The injector implements potential faults, which do not become actual faults if masked. For example, a bit that is already at ‘1’ will mask a fault forcing the bit to ‘1’ as it has no realized effect on the output. Injecting faults is a measure of system reliability, and the results indicate whether the NoC is able to handle faults in data transmission or processing.

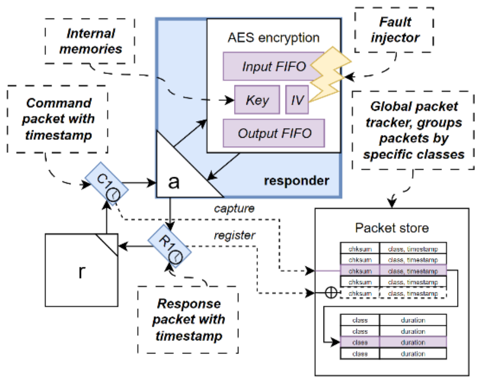

Fig. 5. SystemC simulation environment components. Each packet has timestamps to measure time from packet issue to packet reception at the application level.

For functional verification, the implemented application is AES encryption in CBC mode with 256-bit keys. The commander in the NoC issues the encryption key, an initialization vector, and the data as a continuous stream of command packets. Each redundant module performs independent encryption and writes their results back to the commander, who must vote on the packets at the appropriate network layer. Using AES amplifies faults, as even a single bit flip in the input block creates completely different output strings. Once the application has received validated packets, it checks them against expected output, and counts any errors.

There are a few limitations with the model. First, the extra tiles do not contribute packets to the network, so there is rarely congestion, allowing the latency to be much lower. Additionally, the router models do not implement virtual channels for each port. Section VI further discusses possible directions with this point. Finally, the commander module does not implement error recovery, and will stop the simulation as soon as it receives the interrupt for an invalid vote. Once again, Section VI discusses this as a possible direction.

V. Results

A. Latency

Fig. 6 displays latency statistics from the simulation. The packets issued from the commander to the responder have the same average latency because the architecture for packet transfer remains the same. However, the throughput, which Fig. 6 does not depict is 3x as high for the custom architecture, as the commander’s adapter can issue 3 command packets in the same clock cycle (CC). In the baseline, the commander serially issued three different command packets at the application layer. The response packets latency from the responders to the commander measure time from issuance to when the CRC check verified the packet in its containing checkpoint. This was much slower at the application level, as it required different sets of buffers. This difference in the average response packet latency explains the total system latency observed. For the baseline, the latency was 180 CC, while the customized version was 56 CC, less by a factor of 3.1.

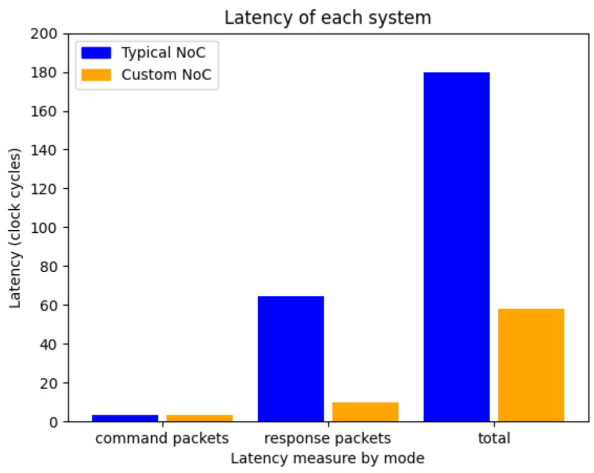

Fig. 6. Latency of packets and the system. The packet latency shows an average over all transmitted packets from application initiation to application reception. The total latency displays the end-to-end latency of the whole application.

B. *Memory and Port Lists*

Because the router crossbars must now consider seven input directions while keeping five output directions, the amount of memory for virtual channels increases by a factor of 12/10. The memory required for the CRC buffers is the same in both solutions. In the baseline model, this memory was in the application layer, whereas in the custom approach, the adapter contained this memory. This is an important consideration because the adapters grow in area. While it is a small increase, it is important and reduces the available area for corresponding tiles. One other benefit of implementing this solution in the adapter instead of the router is that the feature is optional. Not all tiles need to have adapters with the customized adapter. In terms of the router and number of ports, the SystemC model implemented a module which was completely functional when only five of the seven input ports were active. Hence, the same router design applies to both cases. Fig. 7 depicts the new interface between custom adapter and custom router.

VI. Conclusion

This project was able to accomplish a reliable design of a Network-on-Chip (NoC) architecture using custom adapters. In examining the network stack, inserting these mechanisms in the link control layer offers a 3x speedup over leaving the mechanisms for the application. This does introduce an overhead in the memory and port list requirements on the adapter, to which this paper proposed mitigations. The design is able to solve application-level redundancy, but does not implement a rigorous model of NoC virtual channels or error recovery. The use of virtual channels can enhance redundant transmissions, especially in a congested system, by providing guaranteed service for the safety critical applications [3]. Using the modeled interface between the adapter and the application tile easily allows extension to error recovery. The interrupt signal makes the local tile aware of an invalid vote, and how the application handles that is the study of other works.

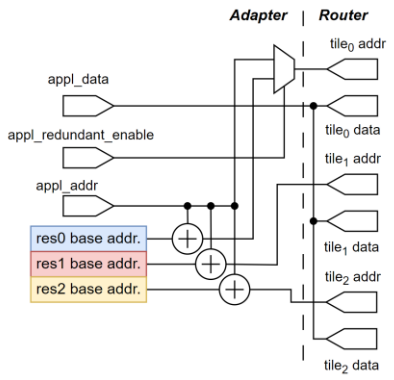

Fig. 7. Port list with minimal wires. The adapter can output one set of wires for the data packet, which fan out to the three readable tile interfaces on the router. There must be partially separated link control port groups, as the destination addresses differ from port to port.

As discussed in the related work section, hardware redundancy is also a relevant topic. In combining this design with the fault-tolerant routing algorithms or data paths, NoC architectures can become incredibly robust. Hence, reliability can be less of a hurdle for implementing safety-critical applications on multiprocessor systems-on-chip (MPSoC). As a result, such applications, such as computer systems in the aerospace industry, can fully benefit from the performance benefits that MPSoCs, namely NoCs, provide while satisfying fault-tolerant regulations and constraints.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," presented at the Proceedings of the 38th annual Design Automation Conference, Las Vegas, Nevada, USA, 2001.
[2] X. Li, G. Yan, and C. Liu, "Fault-Tolerant Network-On-Chip," in *Built-in Fault-Tolerant Computing Paradigm for Resilient Large-Scale Chip Design: A Self-Test, Self-Diagnosis, and Self-Repair-Based Approach*. Singapore: Springer Nature Singapore, 2023, pp. 169-241.
[3] A. Weichslgartner, S. Wildermann, M. Glaß, and J. Teich, "Invasive Computing," in *Invasive Computing for Mapping Parallel Programs to Many-Core Architectures*: Springer Singapore, 2018, pp. 9-43.
[4] J. Caplan, M. I. Mera, P. Milder, and B. H. Meyer, "Trade-offs in execution signature compression for reliable processor systems," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 24-28 March 2014 2014, pp. 1-6, doi: 10.7873/DATE.2014.106.
[5] B. H. Meyer, B. H. Calhoun, J. Lach, and K. Skadron, "Cost-effective safety and fault localization using distributed temporal redundancy," in *2011 Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 9-14 Oct. 2011 2011, pp. 125-134, doi: 10.1145/2038698.2038719.
[6] B.-A. Tabacaru, M. Chaari, W. Ecker, and T. Kruse, "Runtime Fault-Injection Tool for Executable SystemC Models," 2014.