Implementation of a Loosely-Coupled Lockstep Approach in the Xilinx Zynq-7000 All Programmable SoC™ for High Consequence Applications

Ryan D. Kral, Johanna S. Chong, Anita L. Schreiber
Sandia National Laboratories
Albuquerque, New Mexico, USA
SAND2017-0395 C
Contact Author Email: rdkral@sandia.gov

Abstract: *For high consequence applications requiring information assurance, the architecture of the Xilinx Zynq-7000 All Programmable SoC™ precludes a tightly-coupled lockstep approach between the two processors. Therefore, a loosely-coupled lockstep approach implemented by a transaction checker residing in the Programmable Logic portion of the Zynq device will be discussed along with implementation results and latency measurements.*

Keywords: embedded systems; loosely-coupled lockstep; lockstep; high consequence; Zynq; Processing System; Programmable Logic; Transaction Checker Architecture; interrupt handling; AXI; comparator; event bus.

Introduction

In high consequence applications, information assurance is a vital practice applied to ensure that valid data is used only by the specific targets for which it is intended. Sending invalid data to a correct target or valid data to an incorrect target results in an information assurance violation. This violation results in an error, which can potentially lead to catastrophic failures in high consequence applications.

Multi-processor lockstep systems can be used in an effort to increase information assurance in high consequence applications. A tightly-coupled lockstep system is one in which two or more processors synchronously run identical code. Before each instruction is executed, a hardware comparator is used to determine that each processor is in fact running the same instruction. In this way, bit flips, single processor hardware failures, and other errors are identified immediately, and the application can be halted or reset before any incorrect data is used.

In systems which lack the necessary hardware, or do not require the level of rigor that a tightly-coupled lockstep architecture provides, a loosely-coupled lockstep design can be used. In this case, processor data is only compared before it leaves or enters the immediate system. This does not ensure that each instruction is executed properly, but errors are caught before they are allowed to leave the processors and propagate to other devices. This can also lead to performance boosts over tightly-coupled lockstep designs because fewer comparisons are performed.

The Xilinx Zynq-7000 All Programmable SoC™ precludes a tightly-coupled lockstep approach between its processors because no hardware comparator is built into the device. However, a loosely-coupled lockstep approach is achievable through the creation and use of a transaction checker residing in the Programmable Logic portion of the Zynq device. The transaction checker can be combined with specific processor configurations, interrupt handling schemes, and communication channels to create the Transaction Checker Architecture which allows the Zynq to operate as a loosely-coupled lockstep device. The design, implementation details, performance results, and future work of the Transaction Checker Architecture will be discussed in the remainder of this paper.

Transaction Checker Architecture

Implementation of a loosely-coupled lockstep design using the Xilinx Zynq-7000 All Programmable SoC™ is made possible through the use of ARM® Cortex™-A9 MPCore™ Asymmetric Multiprocessing; processor configurations utilizing the On-Chip Memory, L2 Cache, Memory Management Unit, and Snoop Control Unit; dedicated Programmable Logic IP; the Processor Event Bus; various Advanced eXtensible Interface (AXI) bus ports; the AXI Interrupt Controller Programmable Logic IP; and the Generic Interrupt Controller. These elements and their specific configurations make up the Transaction Checker Architecture.

Xilinx Zynq-7000 All Programmable SoC: The Zynq device has several key features which are required for a hardware implementation of this loosely-coupled lockstep architecture. Most importantly, the Zynq contains a dual-core ARM Cortex-A9 MPCore based Processing System (PS) and Programmable Logic (PL) portions. These features allow for two processors to run identical code in the PS with a comparator located in the PL. In addition, there are a number of communication channels which exist so that the PS and PL can easily interface with one another; these include AXI bus interfaces and discrete signals [1].

Asymmetric Multiprocessing (AMP): In order to run an application using a loosely-coupled lockstep architecture, two processors must be configured to execute identical code. To achieve this, the Zynq was configured for AMP with each processor running separate, identical copies of the application code. During the boot sequence, the application code copies are loaded into separate memories. Following the boot sequence, each processor begins executing its respective code copy.

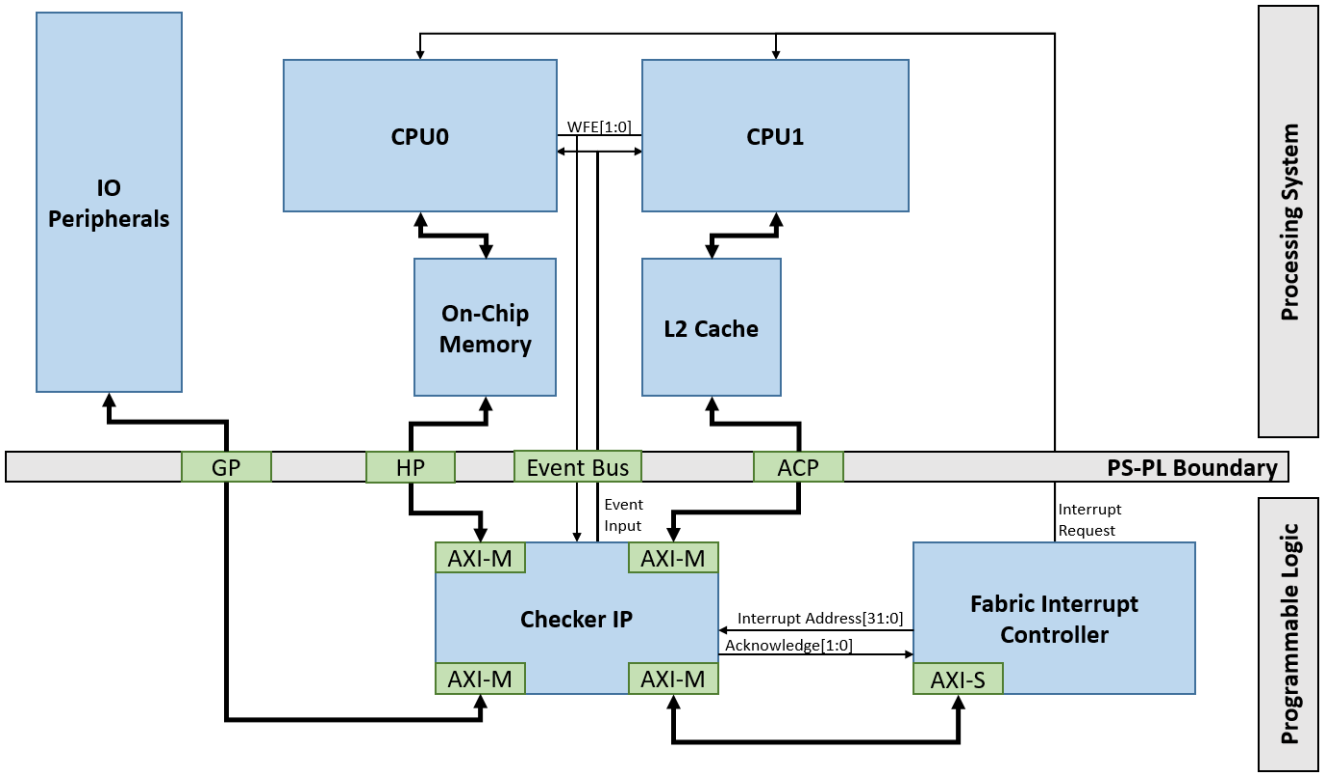

Figure 1: The Zynq-based Transaction Checker Architecture block diagram. The Zynq Processing System is described in the upper portion and the Programmable Logic is shown in the lower portion.

Processor Memory Management: In order to help ensure that each processor executes independently of the other, separate memories and memory management must be used in the Zynq PS. The code for one processor is run from On-Chip Memory (OCM). The code for the second processor is run from a locked L2 Cache (L2C) from which instructions cannot be evicted.

The PS Memory Management Units (MMUs) and Snoop Control Unit (SCU) are used to limit the range of addresses and ports that each processor can access. If a processor attempts to access a blocked address or port, an abort error occurs, and the application is halted [1]. This strategy ensures that a processor cannot see the memory of the other processor.

Transaction Checker Programmable Logic IP: Each peripheral access that the processors attempt to perform must be compared between processors to create a loosely-coupled lockstep system. To perform this comparison on data from two separate memories, a comparator is needed. The Transaction Checker PL IP (which will be referred to as the Checker IP moving forward) was developed to act as a comparator in addition to several other functions.

The primary functions of the Checker IP are the following:

- Validate that the peripheral accesses each processor intends to perform are identical to one another
- Read from or write to the target peripheral in the way described by the processors
- Allow for interactions with a PL-based interrupt controller, referred to as the Fabric Interrupt Controller (FIC)
- Detect and report any error conditions and cause an alarm if necessary

While the comparator function of the Checker IP is clearly necessary, the other three functions are also important. The Checker IP must be the sole initiator of any peripheral reads or writes so that no single processor can access peripherals on its own. Interrupts will be partially handled in the PL, so the Checker IP must also be able to configure the FIC, fetch interrupt addresses, and acknowledge the FIC on behalf of the processors. Finally, the checker must detect errors and prevent them from propagating to other devices by either halting or resetting the system.

Shadow Registers: The processors use shadow registers to communicate the form of read and write peripheral accesses to the Checker IP. Shadow registers are 32-bit data words located at constant memory addresses. Physical to virtual address mapping is used so that each processor accesses the same virtual addresses for their shadow registers. However, the Checker IP uses separate physical addresses to access the shadow registers from OCM and L2C.

Each processor has a set of 20 shadow registers: one for status, one for address, two for control, and 16 for data. The status register provides a way for the Checker IP to give the processors successful or failed transaction codes. The 32-bit address shadow register allows the processors to specify the address of the pending read or write. The control registers allow the processors to specify things such as read or write type, transaction length, AXI burst type, etc. Finally, the data registers are used to send and receive data. There are sixteen 32-bit data registers due to the constraints imposed from the combination of AXI3 protocol [2] and the GP AXI ports [1].

Processor Event Bus: The processor event bus is used to notify the Checker IP when the processors have requested a transaction that needs to be compared. This prevents the Checker IP from continually reading the shadow registers and comparing them between processors. The event bus contains a handful of discrete signals that travel between the PS and PL; in particular, the wait for event (WFE) signals and the event input signal were utilized. Each processor controls a WFE signal which is low when the processor is active and high when the processor is waiting for an event [1]. When a processor has configured its shadow registers and is ready for a peripheral transaction, it begins waiting for an event which halts its execution. It then resumes execution on the receipt of a toggled event input signal [1].

In the Transaction Checker Architecture, the WFE signals are used to indicate to the Checker IP that each processor is halted and ready for a transaction. The shadow registers are then read by the Checker IP, and the transaction begins. Upon completion of a full transaction, the Checker IP toggles the event input signal which causes each processor to resume execution.

AXI Communication Ports: The Zynq has a number of AXI3 interfaces for burst data transfer at the PS-PL boundary. The Transaction Checker Architecture requires three access points at the PS-PL boundary for moving data between the Checker IP and the OCM, L2C, and IO Peripherals (IOP). The 64-bit Accelerator Coherency Port (ACP) provides a low latency path between L2C and a PL master. A 64-bit High Performance (HP) slave port is used as the interface to the OCM. By using 64-bit wide ports, AXI bursts to the processors are as wide as possible. However, accessing the IOP from the PL is limited to a 32-bit General Purpose (GP) port [1]. This means that AXI bursts to peripherals are limited to 32-bit wide transfers.

In total, the Checker IP is equipped with two 64-bit masters to communicate with two sets of shadow registers, one 32-bit master to transfer data to and from the IOP, and one 32-bit master to communicate with the FIC and any other peripherals which reside in the PL.

IO Peripherals: The Zynq IOP contains a number of peripheral registers which can be used to configure General Purpose IO (GPIO), Universal Asynchronous Receiver/Transmitter (UART) controllers, Controller Area Network (CAN) controllers, and many others. Configuration of these peripherals is handled entirely through register writes and reads which are routed exclusively through the Checker IP. From the Checker IP, these accesses travel through the GP AXI slave ports, are routed through various interconnects, and terminate at the appropriate end peripherals.

Interrupt Handling: The Zynq architecture utilizes the ARM Generic Interrupt Controller (GIC) for interrupt handling. The GIC allows either processor to service Shared Peripheral Interrupts (SPIs) in an AMP system [1]. While this facilitates a distribution of the workload, it is necessary to bypass this capability in a loosely-coupled lockstep design. If only one processor services an interrupt, the two processors will no longer be running identical code. This violates the key principle of a lockstep design. To avoid this, Private Peripheral Interrupts (PPIs) are used. Each processor in the Zynq has a set of PPIs that are leveraged to ensure that both processors receive and service the same interrupts and that data transfers resulting from interrupts are compared in the Checker IP before entering or leaving the chip.

Disabling the SPI capability eliminates the processors' direct access to the IOP interrupts. As a result, the peripheral interrupt lines are routed into the PL portion of the Zynq, and an FIC is needed to act on the rerouted interrupt lines. It receives the interrupts and forwards them on to the processors as PPIs. This design uses a commercial AXI Interrupt Controller PL IP from Xilinx® as the FIC. It handles up to 32 unique interrupt sources and has a Fast Interrupt Mode in which the IP can send an address over a bus. The interrupt IDs and addresses are programmed by the processors over an AXI-Lite slave interface [3].

Upon receipt of a peripheral interrupt, the FIC generates PPIs to both processors and forwards the interrupt address to the Checker IP. This address specifies the location of the peripheral interrupt handler in memory. The interrupt service routines (ISRs) for the PPIs are handled by the GIC. The ISR of each processor directs it to retrieve the interrupt address through the Checker IP which writes the interrupt address to the processors' shadow registers. The processors then call the appropriate function based on the received interrupt address. At completion, both processors fill their respective shadow registers to indicate to the Checker IP that the interrupt in the FIC can be cleared.

Results

Hardware Testing: The Transaction Checker Architecture was developed and implemented in hardware. Test software was then generated for the architecture which configured GPIO pins and a UART port. Multiple GPIO pins were configured as outputs and used to toggle LEDs in a predefined pattern which was then manually verified. The UART controller was configured for a baud rate of

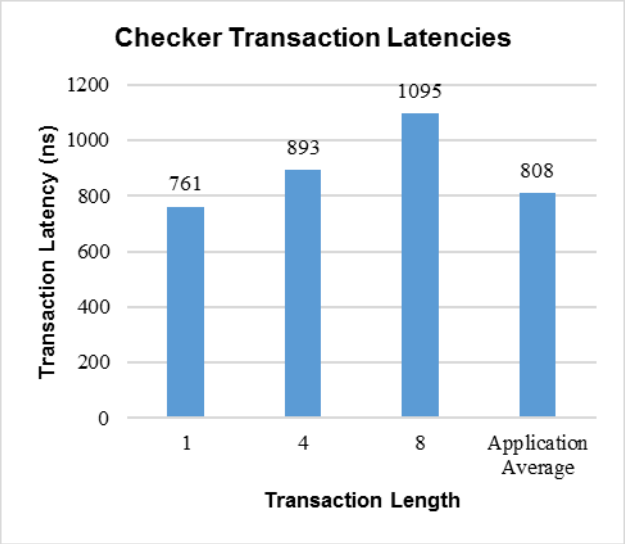

Figure 2: The average checker transaction latencies starting when both processors have WFE set high and ending when the processor event bus event input signal toggles. This is the time that both processors are halted.

115,200, and test messages were sent. The interrupt handling flow was tested using UART interrupts and Virtual IO interrupt events to toggle LEDs. The failed comparison error signal was tested with intentionally bad shadow register values. Overall, the architecture performed as intended with no detriment to application functionality.

Latency Measurements: The latency added due to the Transaction Checker Architecture was analyzed and found to be acceptable. The number of fabric clock cycles when running at 100 MHz was measured. This measurement began when both WFE signals were high and ended when the processor event bus event input signal was toggled. This means that the measured checker transaction latency is equivalent to the time that both processors were halted. The checker transaction latency was calculated based on the number of measured clock cycles and the clock period.

To collect latencies, an application was used which configured GPIO pins for outputs and toggled LEDs in a known pattern. This application was run several times and average measurements were gathered. The results can be seen in Figure 2. For this specific application, the average transaction took 808 ns. When broken down, 1-word transactions took an average of 761 ns, while 8-word transactions took 1095 ns. The timing increase for larger transaction lengths is primarily due to the increased amount of data beats in an AXI burst, as well as an increased AXI response time.

Read or write speeds were also analyzed. A transaction consisting of one 32-bit word takes 761 ns. This translates to a data transfer speed of 42 Mbit/sec. For larger transactions of eight 32-bit words, the transfer speed is 233 Mbit/sec. This architecture does introduce additional delays which are not accounted for in these measurements such as the time for the processors to fill the shadow registers and the synchronization time delay between the two processors. However, these delays are comparable to what would be seen in an application which does not use the Transaction Checker Architecture and would only slightly reduce the transfer speeds. Even with these delays considered, the data transfer speeds are well within reason for a large number of applications, and therefore this architecture can be viewed as a practical loosely-coupled lockstep approach.

Future Work

There are several potential areas for improvement in this design. Continuing efforts to minimize system latency will lead to optimization of both hardware and software techniques. In addition, latency results detailing more complicated peripheral transactions and interrupt events will better characterize the system performance.

There is also further opportunity to expand the Checker IP's AXI3 capabilities. The protocol allows for a number of features which are not yet employed in this design such as the ability to interleave messages and perform concurrent reads and writes. The intelligent use of generics as well as AXI ID and USER bits can make the architecture more flexible to meet various application requirements.

It is also necessary to make the Transaction Checker Architecture more robust. A processor synchronization capability using the Checker IP will be implemented to tackle the potential problems of latency differences between L2C and OCM accesses. A deeper analysis of failure modes and error handling will determine the design's resilience against unexpected and potentially catastrophic events.

Acknowledgements

The authors would like to acknowledge and thank Xilinx, Inc.® for their support during the development of this architecture.

References

1. "Zynq-7000 All Programmable SoC Technical Reference Manual," 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.
2. "AMBA® AXI™ and ACE™ Protocol," 2011. [Online]. Available: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5572/labs/refs/AXI4_specification.pdf.
3. "LogiCORE IP AXI Interrupt Controller (INTC) v4.1, *Product Guide for Vivado Design Suite*," 2013. [Online]. Available: https://www.xilinx.com/support/documentation/ip.../axi_intc/v4.../pg099-axi-intc.pdf