SAND2018-0265C

Configuration of the Xilinx Zynq-7000 All Programmable SoC Memory Resources for Loosely-Coupled Multiprocessor Lockstep Applications

Ryan D. Kral
Sandia National Laboratories
Albuquerque, USA

James Groening
Sandia National Laboratories
Albuquerque, USA

***Abstract*—For high consequence applications, a loosely-coupled lockstep approach can be implemented in the Xilinx Zynq-7000 All Programmable SoC to provide information assurance. A key element of this approach is the separation of the memory resources allocated to each processor. A separation method utilizing On-Chip Memory, L2 cache locking, the Snoop Control Unit, and the Memory Management Unit is presented.**

Keywords—embedded systems; loosely-coupled lockstep; lockstep; high consequence; Zynq; Processing System; Programmable Logic; comparator

I. Introduction

Information assurance is a critical responsibility of high consequence systems. These systems must behave exactly as intended, and errors must be identified and addressed before they propagate.

The use of a multi-processor lockstep design is one way to increase the information assurance of a system. In a typical, tightly-coupled lockstep design, two or more independent processors attempt to run identical instructions which are compared before they are executed. In the case of an instruction mismatch between processors, voting schemes or deterministic error handlers are implemented to prevent the execution of incorrect instructions from propagating in unknown ways.

The Xilinx® Zynq® UltraScale+™ MPSoC is one example of a system on a chip (SoC) which contains two processors configured to run in tightly-coupled lockstep. However, the popular Xilinx Zynq-7000 All Programmable SoC contains two processors but no instruction comparator. This allows for parallel execution of code, but there is no mechanism to keep the processors in tightly-coupled lockstep.

For applications which use the Zynq-7000 SoC and require information assurance, a loosely-coupled lockstep architecture, named the Transaction Checker Architecture, has been designed and implemented. This architecture leverages the programmable logic portion of the device to instantiate the Checker IP which verifies that I/O transactions between the two processors are identical. Errors can still propagate internally within this architecture, but they will be caught before they leave the SoC.

The Transaction Checker Architecture is broken down into two main components. First, there is the programmable logic component which contains the Checker IP, an interrupt handling scheme, and communication channels to and from the peripherals and processors. This component of the architecture is described in detail in "Implementation of a Loosely-Coupled Lockstep Approach in the Xilinx Zynq-7000 All Programmable SoC for High Consequence Applications" [1]. The second key component of this architecture is the configuration of the processing system. The primary goal of the processing system configuration is to separate one processor and its memory from the other processor and its corresponding memory. Without separation, errors could potentially occur in both processors in the same way and they would not be caught by the comparator.

This paper will cover the configuration necessary for the Transaction Checker Architecture to succeed on the Zynq-7000 SoC. First, a brief description of the Zynq-7000 device will be given. Next, the Transaction Checker Architecture will be covered at a high level. Then, detailed descriptions of the settings for various processing system resources will be given. Finally, the boot process necessary to run in loosely-coupled lockstep will be described.

II. Key Zynq-7000 SoC Resources for the Transaction Checker Architecture

The Xilinx Zynq-7000 All Programmable SoC contains several resources which are utilized in this loosely-coupled lockstep architecture. It provides a dual-core ARM® Cortex™-A9 MPCore based processing system (PS) which can be configured for asymmetric multiprocessing (AMP). AMP allows the two processors to execute independently of one another. Each processor has its own L1 instruction and data caches. Each processor also contains its own Memory Management Unit (MMU) which performs virtual to physical address translations. The two processors share 256 KB of On-Chip Memory (OCM), and 512 KB of lockable L2 cache which can be used as instruction and data memories for the processors. The PS contains a single Snoop Control Unit (SCU) which performs address filtering between the OCM and L2 cache and typically handles cache coherency between the processors. The Zynq-7000 SoC also contains FPGA fabric and multiple Advanced eXtensible Interface (AXI) bus

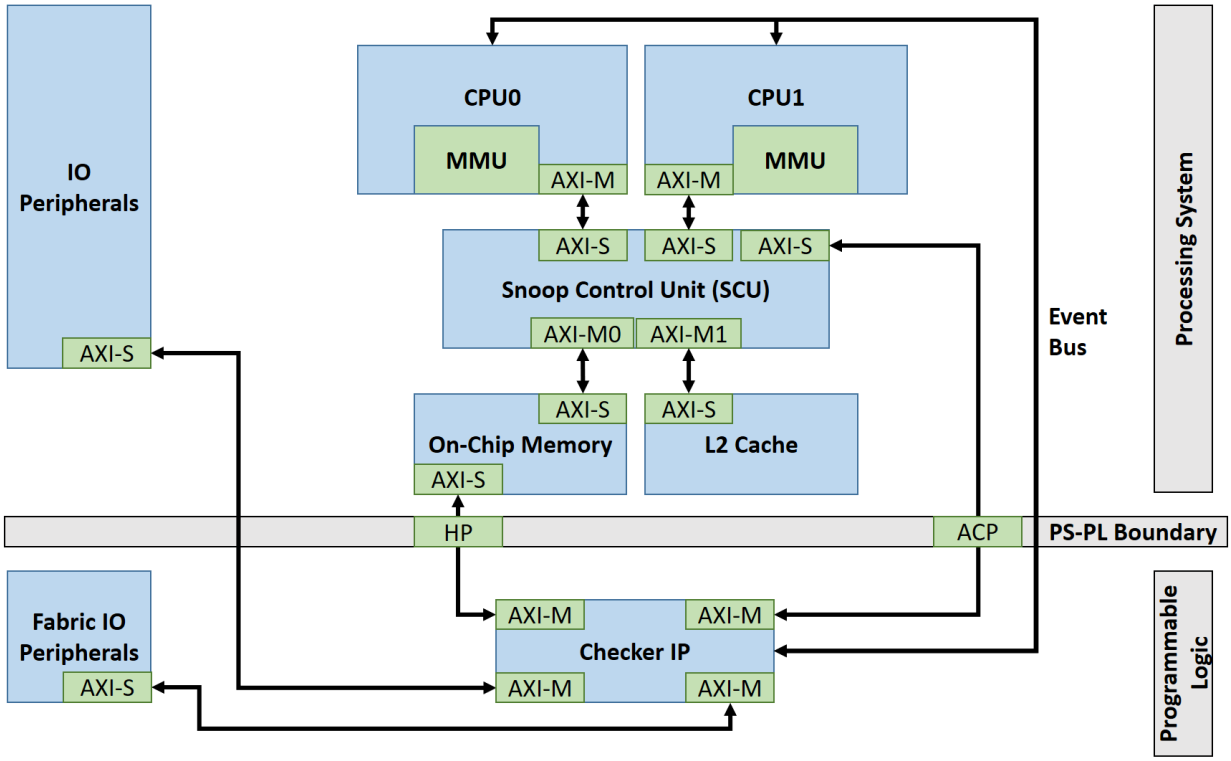

Figure 1: Diagram detailing the processing system components and routing within the Transaction Checker Architecture.

interfaces which allow the PS and PL to communicate with one another.

III. Transaction Checker Architecture Description

For a successful loosely-coupled lockstep design, both CPUs must execute identical application code from memories which are separate from one another. In the Transaction Checker Architecture, CPU 0 will have its instructions and data stored exclusively in OCM while CPU 1 will have its instructions and data stored exclusively in a locked L2 cache. Because the L2 cache is being used atypically as a RAM, the L1 caches will not be used, and cache coherency will not be allowed.

In this architecture, the virtual to physical address mappings for CPU 0 and CPU 1 must differ so that the processors execute instructions and access data with identical virtual addresses, but different physical addresses. The MMUs ensure that CPU 0 only attempts to access OCM physical addresses and CPU 1 only attempts to access physical addresses which have been loaded into the L2 cache. As an additional layer of separation, the SCU is configured to perform address filtering such that CPU 0 can only access the port which leads to OCM, and CPU 1 can only access the port which leads to the L2 cache.

When the CPUs wish to perform an I/O access, they write to their respective memories and signal to the Transaction Checker PL IP that it needs to perform a transaction. As seen in Figure 1, the Checker IP accesses the transaction data from OCM directly, and accesses the data in the L2 cache by going through the SCU. The Checker IP verifies that both CPUs have separately arrived at the same I/O access and performs the read or write. If it detects a mismatch between the processors, it flags an error which leads to a system reset.

The aforementioned component settings must first be configured with boot software which cannot be executed in a loosely-coupled lockstep manner. Instead, separate portions of the boot software are executed by either CPU 0 or CPU 1 to copy application code into the appropriate memories, lock the L2 cache, and configure the MMUs and SCU. Only after all settings are configured can the application software truly run in loosely-coupled lockstep within the Transaction Checker Architecture.

IV. Application Memories

Typical applications which run on the Zynq-7000 SoC utilize the OCM which is internal to the device, an external DDR memory, or a combination of both to store application instructions and data. In the Transaction Checker Architecture, the DDR is not an option for application memory as it is an external peripheral and must only be accessed following a comparison between the two processors. This makes using the L2 cache as RAM necessary so that two internal memories can store application code and data for the two separate processors. The configurations of the internal OCM and L2 cache to enable this architecture are discussed in the following sections.

A. On-Chip Memory

In the Zynq-7000 SoC, the OCM contains 256 KB of RAM and 128 KB of unmodifiable BootROM which contains the code that is executed when power is applied to the device

[2(TRM)]. The RAM portion of OCM is split into four 64 KB sections which can each be mapped to within either the lowest or highest 256 KB of the Zynq-7000 address map. All four OCM sections must be remapped to contiguous addresses to store the application code for CPU 0. In the default flow of the Zynq, the unmodifiable BootROM hands off execution to a section of OCM mapped to a low address range. This architecture then remaps the rest of the OCM sections to the low address range as well.

As shown in Figure 1, OCM is accessible by both CPUs through AXI Master port 0 in the SCU [2(TRM)]. In the Transaction Checker Architecture, only CPU 0 will access this port. Figure 1 also shows that OCM is accessible from the PL via the Accelerator Coherency Port (ACP), or the High-Performance (HP) ports [2(TRM)]. The Checker IP uses an HP port to access the transaction information in the OCM.

B. L2 Cache

The processors in the Zynq-7000 SoC share a unified 512 KB, eight way, set associative L2 cache. Each way is 64 KB in size [2(TRM)]. The L2 cache controller allows lockdown by ways within the cache through its lockdown register, which prevents entries from being evicted. Locking code and data in the L2 cache allows the cache to be used as if it were an on-chip memory.

Before the L2 cache is locked, it must first be loaded. To load the L2 cache, the application must be placed in cacheable memory. In the Transaction Checker Architecture, the application is copied into a portion of DDR which is configured by the MMU to be inner and outer cacheable with no writethrough. This memory must be both inner and outer cacheable so that the program will be cached through the L1 cache and into the L2 cache. The memory region should be marked as no writethrough so that once the application is loaded and locked, no L2 cache updates will be written out to external DDR. After the application instructions and data are placed into cacheable DDR, they must be preloaded into the cache. Because the cache is not directly addressable, it is loaded by using the ARM PLD prefetch hint instruction. It is important to note that the physical addresses of DDR which were loaded into the L2 cache are the same physical addresses that will be used to access the L2 cache from CPU 1 and the Checker IP.

Once the application is loaded into the L2 cache, the used portions of cache must be locked. Because the OCM is limited to 256 KB of memory, the application should be small enough to fit within 256 KB of L2 cache as well. Thus, only four of the eight ways of the cache need to be locked.

Each of the two processors has its own, separate, 32 KB L1 instruction and data caches [2(TRM)]. In this architecture, the L1 caches are not used. The L1 instruction caches are disabled completely. The L1 data caches cannot be disabled separately from the L2 cache and must therefore remain enabled so that the L2 cache can be used as the application memory for CPU 1. Once the L2 cache is loaded and locked, the memory regions that the CPUs execute from are marked as inner non-cacheable in the MMUs to prevent use of the L1 data caches during loosely-coupled lockstep operation.

V. Enforcing Separation

Now that the memories have been described, the resources and techniques used to keep the memories and CPUs separate can be discussed.

A. Memory Management Unit

The MMU's function is to translate virtual memory addresses to the physical addresses used to access memory resources such as DDR, OCM, or peripheral registers. The Transaction Checker Architecture's loosely-coupled lockstep approach takes advantage of the MMUs to allow identical code to be written for both processors. The code utilizes virtual addresses, while the processors execute from separate physical memories.

The virtual to physical translation process uses translation tables stored in memory. These translation tables can contain entries for the entire addressable memory range. Each entry specifies how a range of virtual memory is mapped to physical memory. These entries can be specified for memory ranges sized from 4 KB to 16 MB.

The MMU contains Translation Look-aside Buffers (TLBs) which store recently used translation table entries. This acts as a cache for the translation table. TLBs can be locked, just like cache lines. A locked TLB entry will never be evicted. Locking the TLBs allows for a limited set of memory translations to be located within the MMU, which removes the need to fetch translation table entries from external memories. If the locked TLB entries cover the address range needed by a program, then an external translation table is not needed. Without a translation table, any access to memory outside of the ranges specified in the TLBs will lead to a memory access exception.

In the Transaction Checker Architecture, the MMUs and their TLBs are configured during the boot process. CPU 0's MMU is configured so that all virtual memory addresses used in the program are translated to OCM. Similarly, CPU 1's MMU is configured so that all virtual memory addresses used in the program are translated to the L2 cache. These translation table entries are loaded and locked into the TLBs, and no translation tables are stored or used.

B. Snoop Control Unit

The Snoop Control Unit is typically used to enforce cache coherency between the CPUs in the Zynq-7000 SoC. For applications to run in loosely-coupled lockstep, the L1 caches are not used and the L2 cache must remain inaccessible by CPU 0. This means that the cache coherency features of the SCU must be disabled. This is done by disabling symmetric multiprocessing in the system. Instead, AMP must be used in this architecture.

In addition to cache coherency, the SCU handles address filtering between the OCM and L2 cache. Based on the physical addresses that the CPUs or ACP attempt to access, the SCU filters the bus transactions to either the OCM or L2 cache. Figure 2 shows the paths through the SCU that the CPUs and ACP can take after the SCU and MMUs are configured. CPU 0 has no route to the L2 cache. CPU 1 and the ACP have no route to OCM.

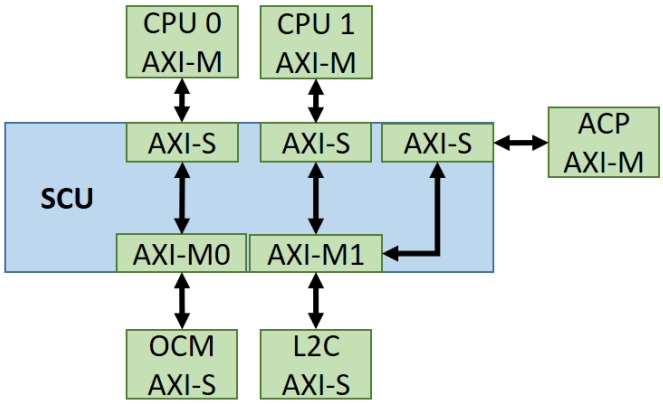

Figure 2: The routing through the Snoop Control Unit after it is configured for the Transaction Checker Architecture.

VI. The Boot Flow

Now that the various component configurations have been explained, it is necessary to explain the order and means by which they happen. The boot process begins when the Zynq-7000 SoC default BootROM hands off execution to a custom First Stage Boot Loader (FSBL) running on CPU 0 from OCM.

The FSBL begins by remapping all sections of OCM to low addresses. It then loads the bitstream from QSPI flash and copies the Second Stage Boot Loader (SSBL) from QSPI into external DDR. CPU 0 then jumps to and begins executing the SSBL.

In a typical boot process, an SSBL is optional. It is necessary in this architecture for two main reasons:

1. The SSBL is responsible for copying the application code from QSPI to OCM. Because the FSBL runs from OCM, it cannot be overwritten until execution is switched to a different memory.
2. The SSBL is necessary to load and lock the application into the L2 cache. Because there is no direct path to load the L2 cache from OCM, the SSBL running from DDR is used to perform the loading and locking.

After copying the application code from QSPI to OCM and loading and locking it into the L2 cache, the SSBL resets CPU 1. CPU 0 then spins in an idle loop until it is reset later in the process.

Whenever a processor is reset, it begins executing from physical address 0x0 in OCM with its MMU disabled. This means that although the OCM and L2 cache are loaded with the same application code, the initial portion of application startup code will only ever run from OCM, regardless of which processor is executing. After coming out of reset, CPU 1 runs the application startup code to configure the SCU and its MMU. It ensures that cache coherency is disabled and address filtering is enabled in the SCU. CPU 1 then loads and locks its MMU with the translation table entry necessary to continue executing from OCM as well as the translation table entry needed for the application code to run from the L2 cache. Once the TLB entries are locked, CPU 1 enables its MMU and jumps to the virtual application start address. Because the MMU is now enabled, this jump switches execution on CPU 1 to the L2 cache.

Once executing from the L2 cache, CPU 1 resets CPU 0 so that it will begin executing its startup code in OCM. CPU 1 then removes the translation table entry for the OCM startup section from its MMU. CPU 1 is now at the start of the application.

When CPU 0 comes out of reset, it begins to execute its MMU startup code in parallel with the final boot steps of CPU 1 which run from the L2 cache. CPU 0 loads and locks its MMU with the translation table entry needed to run from OCM. CPU 0 then enables its MMU and jumps to the virtual application start address. This will keep CPU 0 executing from OCM and will bring it to the start of the application.

At this point in time, both CPUs are ready to execute identical application code from separate memories. The Checker IP identifies that both CPUs have been configured and it releases the processors to run in loosely-coupled lockstep.

Until SCU address filtering and both MMUs have been configured, there is a potential for errors which may cause undesired access to peripherals. To reduce this potential, the MMU and SCU settings can be configured earlier, during the FSBL. This will reduce, but not eliminate, the time that the CPUs have access to peripherals without going through a comparison in the Checker IP.

	CPU 0	CPU 1
FSBL (OCM)	• Remap OCM to low addresses • Load bitstream from QSPI • Copy SSBL from QSPI to DDR	
SSBL (DDR)	• Copy application from QSPI to OCM • Copy application into DDR • Load and lock application into the L2 cache • Reset CPU 1	
App Startup (OCM/L2 Cache)	• Load and lock CPU 0 MMU TLBs • Enable MMU • Jump to application	• Configure the SCU • Load and lock CPU 1 MMU TLBs • Enable MMU • Switch execution to the L2 cache • Reset CPU 0 • Finalize CPU 1 MMU TLBs • Jump to application

Figure 3: A chart which describes the boot steps necessary before the Zynq-7000 SoC processors can run in loosely-coupled lockstep. The steps occur in order from top to bottom. Steps at the same level are performed in parallel.

VII. Future Work

There are many possible future extensions to the Transaction Checker Architecture. However, the primary

concern is that of application size. In the current architecture, an application is limited to 256 KB of memory as that is the size of the OCM. This is a stringent requirement which leads to tradeoffs between information assurance and features in some high consequence systems. For systems which require additional application memory, two potential solutions are under investigation:

1. Block RAM in the FPGA fabric can be used for storage of additional application code and data for CPU 0. A memory swapping mechanism can be added to the PL which would enable an increased application memory space. If 256 KB of block RAM were allocated to CPU 0 in the PL, CPU 1 could be given access to all 512 KB of L2 cache. This would double the potential application size which could be run in loosely-coupled lockstep.
2. The unused 256 KB of L2 cache could be split in half and allocated between CPU 0 and CPU 1. This would reduce the physical separation benefits gained from SCU address filtering as both CPUs would now share AXI Master port 1 of the SCU. However, MMU translation table entries could still be utilized to ensure the separation of memory regions. This approach would increase the potential application size by 150%.

VIII. Summary

The Transaction Checker Architecture provides high consequence systems with an approach to increase information assurance. It allows applications which run on the Zynq-7000 SoC to run in loosely-coupled lockstep with two independent processors executing identical code from two separate memories. The Checker IP in the PL prevents unique errors from propagating outside of the device. The memory configurations, separation techniques, and the boot sequence described in this paper prove that increasing information assurance in high consequence systems an achievable task.

References

[1] R. Kral, J. Chong, and A. Schreiber, “Implementation of a Loosely-Coupled Lockstep Approach in the Xilinx Zynq-7000 All Programmable SoC for High Consequence Applications,” GOMACTech 2017, Reno, NV.

[2] "Zynq-7000 All Programmable SoC Technical Reference Manual," 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.