FreeBSD on Cavium ThunderX System on a Chip

Zbigniew Bodek, Wojciech Macek
Semihalf,
zbb@{semihalf.com, freebsd.org}, wma@semihalf.com

Abstract

This paper describes the FreeBSD operating system port for the Cavium ThunderX CN88XX System on a Chip. ThunderX is a newly introduced, ARM64 (ARMv8) SoC designed for the high performance and server markets. It is currently the only one in the ARM world to incorporate up to 96 CPU cores in the system along with the whole technology to make it possible. ThunderX is up to date with the latest trends in the computer architecture industry, including those that are relatively new to FreeBSD like SR-IOV (Single Root I/O Virtualization) or completely unique, such as ARM GICv3 and ITS).

The main focus of this article is to provide a bottom-up overview of how the FreeBSD platform support for ThunderX was implemented and what are the benefits and pitfalls of the newly introduced ARMv8 technology in terms of the OS development. The paper also describes the key components of the ThunderX system and explains, how they were supported in FreeBSD. Finally, possible fields of further improvements are pointed out briefly.

1 Introduction

FreeBSD is undoubtedly the most recognizable Unix-like operating system available. One of the main areas of its deployment is server market, which is still dominated by the Intel and AMD-based computers. However, recently the highly successful in the mobile industry ARM architecture is gaining more interest as a foundation for high performance server SoCs. A turning point in that field was the emergence of a 64 bit ARM implementation (ARMv8) including improved technology to obtain insane multi-core capabilities. Thus it is substantial for the FreeBSD community (developers and users) to keep up with the growing interest in ARM-based servers and supply the ecosystem with the ARM64 BSD OS, that will be on par with the available Tier-1 x86 platforms.

The motivation behind the work on ThunderX was driven by the actual market need for the FreeBSD OS on that platform, the aim of which is to become a real alternative for the energy-intensive solutions. ThunderX is also the only ARM-based chip in the world to scale up to 48 CPU cores per socket with the possible dual socket configuration (up to 96 CPUs). With the included hardware accelerators for networking, storage and security, as well as powerful peripheral devices, ThunderX is a perfect ground for the server-oriented OS such as FreeBSD and a great engineering challenge to face for a kernel developer.

The introduced support is based on the foundational work with the emulation as a primary target (ARM Foundation Model). Hence the platform support for ThunderX was partly carried out in parallel to the FreeBSD base system development and occasionally was intertwining with it. ThunderX was also the first hardware platform to switch from the ARM emulator.

The result is a fully functional, though still experimental OS, that supports key chip features such as:

- PCI Express

- GICv3 + ITS
- SMP (single and dual socket)
- SATA
- Virtualized Network Interfaces

The description of the general FreeBSD kernel implementation for ARMv8 architecture is beyond the scope of this paper and is discussed only in respect to the work on ThunderX support.

2 Hardware overview

Contemporary ARM-based chips very much follow the current trends in the computer industry incorporating multiprocessor capabilities, high performance peripheral devices, buses and extensions, as well as various hardware accelerators and virtualization technologies. The 8th architecture revision (ARMv8) moved those concepts to a new level by overcoming previous, architectural limitations. ThunderX is a good representative of the new ARM chips generation as it is the first to utilize majority of the newly introduced features.

2.1 Core complex and CCPI

The heart of the CN88XX SoC is a set of Cavium proprietary ARMv8-compliant CPUs. Each CPU has its own I-cache and D-cache but they share a common, 16MB L2 cache. Single package can contain up to 48 CPUs but the complex can be connected to another CN88XX processor using Cavium Coherent Processor Interconnect fabric (CCPI). All CPUs in the system are cache coherent in respect to L1, L2 cache and DMA accesses. The system coherency capabilities are also extended across CCPI connected entities. Therefore, the dual socket configuration gives a fully coherent system containing 96 ThunderX CPUs.

3 System bootstrap

Typical CN88XX boot scenario starts with the on-chip Boot ROM. This stage is supposed to load a Boot-level 1 Firmware (BL1) from the SPI connected FLASH which will then load further BL2 and BL3 Firmware and finally, the control over the system is passed to the Cavium Unified Extensible Firmware Interface (UEFI) bootloader. At that point ThunderX system and its interfaces are initialized and boot CPU core is in EL2 exception level (Hypervisor). In order to run FreeBSD, ThunderX needs to load the kernel ELF file and supply it with a machine description such as the DTB (Device Tree Blob), available memory regions, as well as the information about the kernel location in DRAM, etc. Hence the next step in the FreeBSD boot process is the native BSD *loader(8)* which in that case is being executed in UEFI runtime environment. *loader(8)* handles kernel acquisition and jumps into its code. ThunderX uses genuine ARM64 *loader(8)* so almost no modifications needed to be made.

4 Early system initialization

The very first kernel code being executed is the one in locore.S. It performs two fundamental actions:

- Puts CPU into a well defined state
- Prepares the execution environment for the C code

The start software usually drops the exception level to EL1 (after performing EL2 specific configuration), creates initial kernel mappings that are required to change the context to the kernel virtual address space, enables Memory Management Unit and jumps to the early machine initialization in C code. ThunderX requires some more settings during that stage than the ARM Foundation Model. This includes:

- Enabling EL1 access to the Generic Interrupt Controller's CPU Interface

By default CPU interface cannot be accessed through System Registers from EL1. Access permission has to be granted while still in EL2.

- Enlargement of the Virtual address space in `TCR_EL1`
  On ThunderX, the physical memory is mapped beyond 512GB. Therefore, if the programmed address space is not big enough, it is impossible to create an identity mapping required to jump from Physical to the Virtual address space.

The changes however are not strictly ThunderX-specific and will apply to any platform with the similar requirements.

5 Interrupts delivery

Exceptions, whose purpose is to indicate to CPU that certain action took place, are called interrupts. There would be no reasonable multithreading OS without interrupts support therefore they are one of the most fundamental elements of the system. Previous generation of ARM processors often used to incorporate so called ARM Generic Interrupt Controller (GIC) or other proprietary implementation.

Exemplary ARM GIC consists of two main components: Distributor and CPU Interfaces. Typically, interrupt lines from the onchip devices are wired to the Distributor interface which then, according to its configuration, routes the interrupt signals to the appropriate CPU interfaces. If the interrupt signalling is enabled, the CPU will receive a notification on the appropriate interrupt line (IRQ or FIQ). Finally, the CPU can acquire the interrupt information such as its number from the CPU Interface registers and change pending state of the interrupt. This architecture works fine for ARMv6/v7 processors but has some serious limitations in terms of:

- Scalability
  Can route interrupts to up to 8 CPU cores
- Maximum number of interrupts
  Each interrupt requires physical connection to the Distributor
- No Message Signalled Interrupts support
  Cannot use in-band PCI interrupt signalling
- Slow access to the CPU Interface registers
  Each interrupt requires at least few read/write sequences to the memory mapped registers (slow access to device memory and possible TLB misses).

5.1 Generic Interrupt Controller v3

Platforms such as ThunderX require improved interrupts handling to provide better SMP utilization, support for PCIe devices and minimal time penalty per interrupt. These features were introduced with ARM Generic Interrupt Controller v3 in cooperation with Interrupt Translation Service. The contributed work includes full FreeBSD support for ARM GICv3 and ITS along with the ThunderX specific quirks but excluding virtualization extensions.
This paper describes the support for the crucial GIC components only and does not cover the implementation of the machine dependent part of the interrupts handling code that needed to be redesigned for the purpose of this port.

5.1.1 Affinity-based routing

Unlike earlier GIC architectures, GICv3 incorporates additional, third component in form of Re-Distributors that are memory mapped entities, associated with every CPU in the system. However, CPU Interface can be optionally accessed through CPU's System Registers to speed up interrupts handling after the core gets the notification. To overcome the interrupt to CPU delivery limitations, the interrupt is now routed based on the Affinity Hierarchy. This means that the interrupt destination is now addressed by the 4-level CPU affinity number in the system. The GICv3 driver configures all SPIs (Shared Peripheral Interrupts) in the global Distributor but PPIs (Private Peripheral Interrupts), SGIs (Software Generated Interrupts) and a new class of LPIs (Local Peripheral Interrupts) are managed through per-CPU Re-Distributors. Each Re-Distributor needs to be enabled or woken up before it can be used. Fortunately GICv3 provides an auto-configuration scheme that allows the OS driver to iterate through device's memory mapped region, match configurator CPU to a correct Re-Distributor (based on the affinity) and perform appropriate actions. Once configured the Re-Distributor interface can be used in a similar manner to the global Distributor.

Inter Processor Interrupts (IPI) can be also delivered based on the CPU Affinity Hierarchy. Because FreeBSD kernel does not enumerate CPUs in SMP according to their hardware affinity it is required to save and match each CPU address with the requested CPU group on every IPI. Software Generated Interrupts (triggered by the write to the CPU Interface register) are used to perform IPI exchange.

5.2 ITS and Message Signalled Interrupts

In a typical scenario the peripheral device requests an interrupt in the Distributor which then forwards it to the appropriate Re-Distributor. Finally the interrupt is signalled to the CPU Interface. The alternative behavior, for which Re-Distributors are used in GICv3, is interrupt routing that bypass the Distributor. This is used by MSIs and requires ITS assist.

Interrupt Translation Service is a GICv3 extension that routes LPIs generated by any device that can send Message Signalled Interrupts. ITS works closely with PCIe bus and IOMMU because unique device IDs used in the interrupt translation process are passed within the bus transaction itself. ITS is supported in the FreeBSD by a separate driver. The controller requires some portion of the system memory to operate and this needs to be provided by the OS. Variety of possible ITS implementations imply the existence of the autodetection features that need to be revised when configuring the device. The absolute basic configuration requires:

- Memory for ITS Private Tables
  Translation Tables used by the hardware to match the device's interrupt request with the invariant interrupt number.
- Memory for the Command Queue
  ITS is programmed via this command queue. OS software must set the queue write pointer (GITS_CWRITER) to the start of the queue.
- Memory for the LPI Configuration and Pending Tables
  LPIs pending status is visible through the bitmap in the Pending Tables. Particular LPI can be configured (i.e. masked/unmasked) using array of bytes in the Configuration Table.

Interrupt mappings are created per-device and per-CPU collection (group of CPUs). The implementation defined device identifiers are usually based on the PCIe *Bus:Device:Function* address. For ThunderX this ID is more complicated due to possible multi-node configuration and requires additional *Node:ECAM.number* address. Device IDs differ for the internal and external PCIe units and this needs to be taken into consideration by the ITS code. CPUs on the other hand are matched based on their CPU affinity or Re-Distributor Physical Address.

In order to create a LPI interrupt route the software has to:

1. Reserve an LPI from the LPIs bitmap and enable it
2. Acquire the CPU collection identifier
3. Acquire the Device ID
4. Select an interrupt number to map
5. Send MAPVI command to ITS and wait for its completion

The unique approach presented by the ITS controller is that all MSI-capable devices can use a single memory location to generate an interrupt. Any interrupt ID written to the `GITS_TRANSLATER` register, that has a valid translation for the interrupting device, will result in LPI assertion.

6 SMP

The standard ARMv7-MP specification limits the number of supported CPU cores to 8. Each 4 cores are logically connected to create a cluster. Then, up to two clusters can be combined together using CoreLink interface providing fully coherent 8 CPU core system. Some vendors' implementations of ARMv7 cores can provide up to 16 cluster scalability, which seemed to be theoretical limit for the architecture. The ARMv8 is a huge step forward. The interconnects now are able to address each CPU by its logical location using four-level CPU Affinity Address *(Aff3:Aff2:Aff1:Aff0)*, that allows a significant growth in total core number.

6.1 SMP bring-up

ThunderX CPU cores are managed through a standard ARM Power State Coordination Interface (PSCI). The relevant code was already in the FreeBSD sources and was used as is. The main part of supporting SMP operation on ThunderX was focused on resolving problems in areas such as:

- System and TLB cache management
- IPI and interrupts handling
- Context switching
- Memory ordering
- Operations atomicity

For example, system maintains cache coherence between CPU cores but only within their shareability domain. If the common memory mappings are not marked as shareable in that domain, data copies seen by the CPUs may differ. Similar results can be observed when one CPU modifies shared Translation Table entries and appropriate TLB maintenance operation is not issued and propagated to the secondary cores. In that case CPUs can potentially see different physical frames, with different access permissions at the same Virtual Addresses.

6.2 CCPI and dual socket operation

The ThunderX chips provides ever more sophisticated scalability. Based on Cavium Coherent Processor Interconnect (CCPI), two processors can be lashed together creating a shared memory space. The typical two-socket configuration supports 96 cores and up to 1 TB of system memory. From the operating system perspective, the complete machine looks like it has two separate NUMA (Non-Uniform Memory Access) nodes each made of 48 CPUs and half of the memory. The I/O interfaces (eg. PCIe, SATA) are accessible by both nodes, but it is strongly suggested that all I/O accesses are done by the socket owning the corresponding interface. In that scenario, the whole system performance and peripherals are doubled. The dual socket machine offers twice as much of PCIe links, Ethernet interfaces, etc. and the interrupts are distributed among all CPUs using two separate ITS units. For even bigger workloads, the two-socket Cavium systems can be connected together using a low-latency Ethernet fabric. This allows for hundreds of gigabits per second of aggregated network bandwidth.

7 PCIe

Cavium ThunderX machine provides standardized interface which all peripherals are attached to. The only I/O the CPU provides is a modern PCIe 3.0 bus. All other devices (SATA, Ethernet NICs, etc.) are typical PCIe endpoints which can be easily detected by the operating system and do not require any machine specific resource management code except single PCIe controller driver. ThunderX provides two distinct PCIe interface types: internal and external. The internal one is reduced version of generic PCIe standard providing PCI-like logical access to all peripherals inside ThunderX SoC. The external one, on the other hand, is a fully compatible PCIe 3.0 link allowing easy connection of generic PCIe cards of the size up to x8 lanes.

The ThunderX driver is divided into three parts: generic PCIe hardware accessors, FDT-configured internal PCIe controller and, finally, internal PCIe device representing external PCIe controller. FreeBSD PCI subsystem takes care of almost every aspect of PCI operation, except of some very hardware dependent low-level functionalities. In order to fulfill these requirements, driver needs to provide three things: access to devices configuration space, resource (bus addresses) allocations and interrupt mapping. On CN88XX platform, any access to internal configuration space is done using generic mechanism called ECAM (i.e. all configuration headers of devices are mapped into host memory space; each memory access to that location makes the controller to automagically generate all PCIe requests for the user). External PCIe configuration space is accessed using indirect addressing supported by external controller. Second functionality which driver needs to provide is resource assignment. Fortunately, the Cavium UEFI configures all PCIe tree and fills every BAR with appropriate values. The only thing the driver needs to do is to read those (bus) addresses, mark as used in the Resource Manager (*rman(9)*) and return a result. If some driver is not initialized (this happens for example for NIC's Virtual Functions) then the controller manually allocates necessary bus space and properly configures the BAR. The last thing the driver provides is interrupt mapping. Currently, the only supported interrupt types are MSI or MSI-X, which require some quirks in ITS as well.

Advanced architecture of ThunderX offers a huge amount of internal PCIe devices (more than 200 endpoints). To avoid creating enormous PCIe device trees, these devices are separated into 3 different zones, each is governed by a separate internal PCIe controller. It is also worth noticing that the external PCIe controller is the PCIe device hanging on internal PCIe bus. Although not intuitive, this solution provides an easy way to support various hardware versions of the device. Let's imagine one ThunderX chip has only onw external PCIe available, where another might have tree of them. Using conventional approach, each version of the hardware would require different machine description (i.e. DTB) file to make all the controllers get detected and configured properly. But when the external controller is an internal PCIe device, the number of them is gathered on-the-fly using standard PCIe enumeration technique.

8 VNIC

CN88XX chip has powerful Ethernet capabilities that include 40 Gbps, 20/10 Gbps and 1 Gbps interfaces. ThunderX introduces a flexible and highly programmable design of the network subsystem that allows for efficient hardware resources virtualization and node-to-node connectivity without using external switches. The main objective of the presented work was to provide a basic networking support for all type of available interfaces.

The networking subsystem in ThunderX is partitioned into few, core components:

- BGX - Common Ethernet Interface
- NIC - Network Interface Controller
- TNS - Traffic Network Switch

which in turn implement: MAC layer, Network Interface layer and hardware switching between the mentioned components and other CN88XX devices. Moreover, NIC is a SR-IOV capable device that can incorporate up to 128 Virtual Functions, each providing full network interface features. ThunderX contains two BGX instances that are connected to NIC and its VFs via TNS unit.

The described FreeBSD implementation of the ThunderX networking drivers does not include support for TNS and its features. TNS Bypass logic is used instead, that results in direct connection of BGX units to the corresponding NIC TNS interfaces as well as freedom of Rx/Tx queues assignment to Logical MACs provided by the BGX.

8.1 BGX

Programmable MAC layer is implemented in BGX. It is seen as a normal PCIe endpoint and can be configured without explicit machine description provided. However, BGX-to-Ethernet PHYs association needs to be passed to the driver. Each of two available BGX controllers can provide up to 4 Logical MACs (LMACs) with maximum rate of 10 Gbps or a single 40 Gbps LMAC. Any LMAC can be connected to any of NIC's Virtual Function. In addition, each LMAC can store up to 32 MAC addresses to match against with the intention to drop or accept an incoming packet.

FreeBSD driver configures and enables LMACs and is responsible for MAC setting and link configuration or adjustments. All actions are induced by the NIC driver. During normal BGX driver operation the software polls the MAC layer status to keep NIC Physical Function driver up to date.

8.2 Physical Function

Physical Function driver cooperates with BGXes and TNS, creating a highly programmable network interface. Unlike other popular NIC cards, CN88XX Physical Function does not provide networking capabilities but rather is a resource manager for subordinate Virtual Functions (VFs) and an interface between MAC layer (BGX) and networking interface layer (VNIC). PF supports up to 128 Virtual Functions using PCI SR-IOV technology. The communication between PF and VFs is held using private mailboxes for each VF. The introduced FreeBSD driver uses generic PCI IOV subsystem to create and configure Virtual Functions. It then assigns LMACs and polls for links status using BGX interface. Any changes in MAC layer or configuration requests are signalled to the VFs using Mailbox interrupt. Physical Function receives requests from the VFs in the similar manner.

8.3 Virtual Function

Virtual Functions implement networking capabilities of VNIC. VFs are capable of performing DMA transactions to and from the main memory in order to transfer packet traffic. Each Virtual Function contains a set of queues (QS) that consists of:

- 8 x Completion Queue (CQ)
- 8 x Send Queue (SQ)
- 8 x Receive Queue (RQ)
- 2 x Receive Buffer Ring (RBDR)

The Completion Queue contains data describing all completed actions such as packet send or receive. Other queues are assigned by the Physical Function to the selected CQs, potentially many-to-one. The transmitter side uses SQ to describe the egress data and actions that need to be performed on the packet before it can be sent (for example L3, L4 checksums calculation). Each SQ has assigned a destination Completion Queue. Receive Queues are VNIC's internal structures that describe how to receive packets. They need CQ and RBDR assignments. More than one RQ can be assigned to both CQ and RBDR. RBDRs on the other hand describe free buffers in the main memory that can be used to store received data. Upon packet reception VNIC moves the incoming data to the free buffer provided by the RBDR descriptor and returns its physical addresses to the assigned Completion Queue. An interesting capability of the NIC's Queue Sets is that if the VF is not enabled, its queues can still be used by another, operational VF. This gives a possibility to assign 1 QS to a single VF, all Queue Sets to a single VF or anything in between. However, the presented driver utilizes one Queue Set per VF by default.

9 Future work

9.1 4-level Page Tables

To improve kernel performance, all physical addresses from `DMAP_MIN_PHYSADDR` to `DMAP_MAX_PHYSADDR` are mapped statically into continuous VA region called DMAP (direct map). When the PA space is fragmented (as it is in a dual socket configuration), the VA DMAP range is huge and exceeds the limit what the core is able to address in 3-level Translation Table mode. Semihalf implemented a patch that resolves those issues by creating multiple regions of DMAP range to save space and allow for dual socket device operation. However the lookups in DMAP arrays are likely to reduce performance and hence the 4-level Translation Table solution is the prefered one.

9.2 Multiple ITS support

Currently, only ITS attached to socket-0 is operational but in case of dual socket system, this resource is doubled. FreeBSD/arm64 lacks support for multiple and chained interrupt controllers therefore changes in the interrupts handling code for ARM64 architecture need to be applied.

9.3 I/O performance optimizations

Peripheral device drivers upstreamed to FreeBSD-HEAD are not performance-optimized. It was shown that Ethernet driver does not offer line-rate performance, nor the SATA presents satisfactory IOPS rate. To deploy production-quality system performance must be improved.

10 Acknowledgements

The project is a joint work of Semihalf team, Andrew Turner, ARM Ltd., Cavium and The FreeBSD Foundation.
Work on this paper was sponsored by Semihalf.

11 Availability

The code is publicly available beginning with FreeBSD 11-CURRENT (r289550).