Save power and costs with TI's K2E on-chip networking features

Texas Instruments

David Lide
Processor Software Architect
Texas Instruments

Introduction

The KeyStone™ II generation of multicore System on-Chips (SoCs) from Texas Instruments provides a powerful computing platform for a range of broad market applications including networking, industrial and defense and avionics. These SoCs boast multiple high-frequency ARM® Cortex®-A15 and C66x DSP cores, a range of industry-standard peripherals, a high-speed internal processing bus and up to 18 MB of internal memory (including on-chip caches and SRAM). They also include several co-processors to offload processing in certain key areas from the main cores. One of these key elements is the Network Coprocessor (NetCP), comprised of the packet accelerator (PA) and security accelerator (SA) subsystems, Ethernet switches and supporting queues and direct memory accesses (DMAs). All of these together provide a powerful way to accomplish common networking tasks, many times without involving the host processor at all. This frees the host to have more cycles dedicated to the application. In addition, the on-chip Ethernet switches for 1 Gb and 10 Gb Ethernet help reduce overall board bill of materials (BOM) cost by eliminating the need for a discrete switch. Together, the K2E NetCP and Ethernet switches afford the developer with end application power, space and cost reductions without compromising performance. Furthermore, TI provides various software libraries and Linux patches to simplify development with the network coprocessor. This paper outlines the network subsystem and its associated accelerators on the latest members of the KeyStone II family, AM5K2Ex and 66AK2Ex devices, referred to collectively as K2E.

K2E networking overview

The K2E device family consists of four ARM Cortex-A15 processing cores, located in one cluster with 4 MB of L2 cache and is illustrated in the block diagram, Figure 1 (on the following page). In addition, K2E has a network subsystem illustrated in Figure 2 on page 4. This subsystem is comprised of:

- An integrated, 8 port +1 host port, 1 Gigabit Ethernet switching subsystem,
- A separate 2 port + 1 host port, 10 Gigabit Ethernet switching subsystem,
- A packet processing subsystem comprised of a pipeline of dedicated, programmable packet processors with hardware assist for packet operations such as classification and checksums,
- A security processor that is IPSEC aware,
- A packet-aware direct memory access (DMA) peripheral, and
- A hardware-based queuing subsystem, with
- Dedicated, programmable processors for functions such as quality of service and traffic shaping.

Tying these hardware components together are ARM software kernel drivers and user space libraries that enable high throughput, low-latency packet-processing applications. The ARM cluster caches and DMAs are kept coherent via hardware

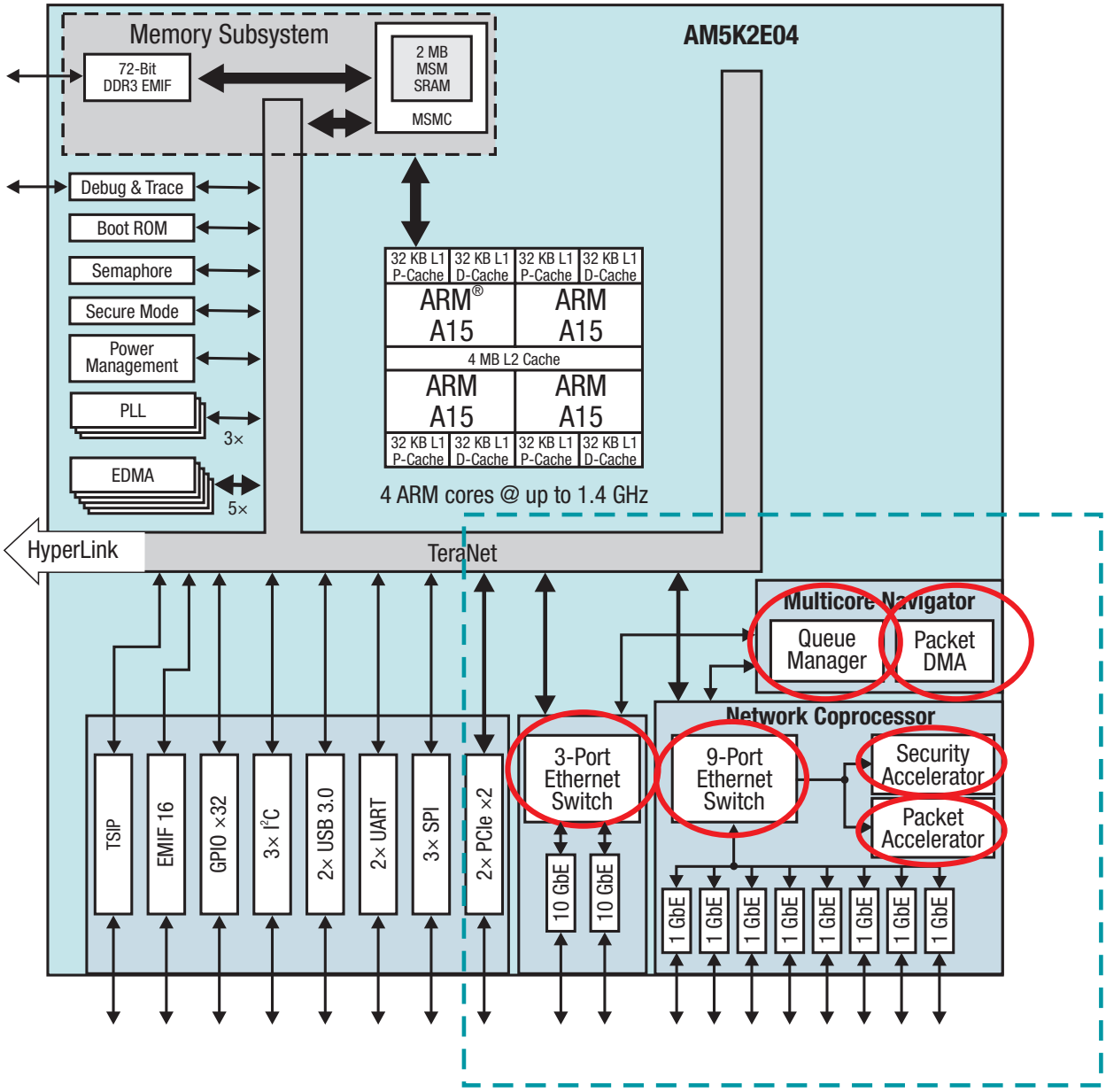

Figure 1. K2E (AM5K2E04) block diagram

making for very efficient packet input to and output from the ARM cores.

Ethernet switch blocks

There are two Ethernet switch blocks in K2E. The integrated switch subsystem in K2E is a full-featured, 801.1d compliant, Ethernet bridge that supports eight 1 Gig Ethernet ports. All Ethernet ports can also be independently used as normal Ethernet ports without switching. The 9-port switch subsystem has numerous features including:

- Line rate switching for 8 external 1 Gig ports and an internal, 8 Gig "host port"
- VLAN and DSCP aware switching
- 8-KByte look-up table for learned and provisioned MAC addresses, VLANs

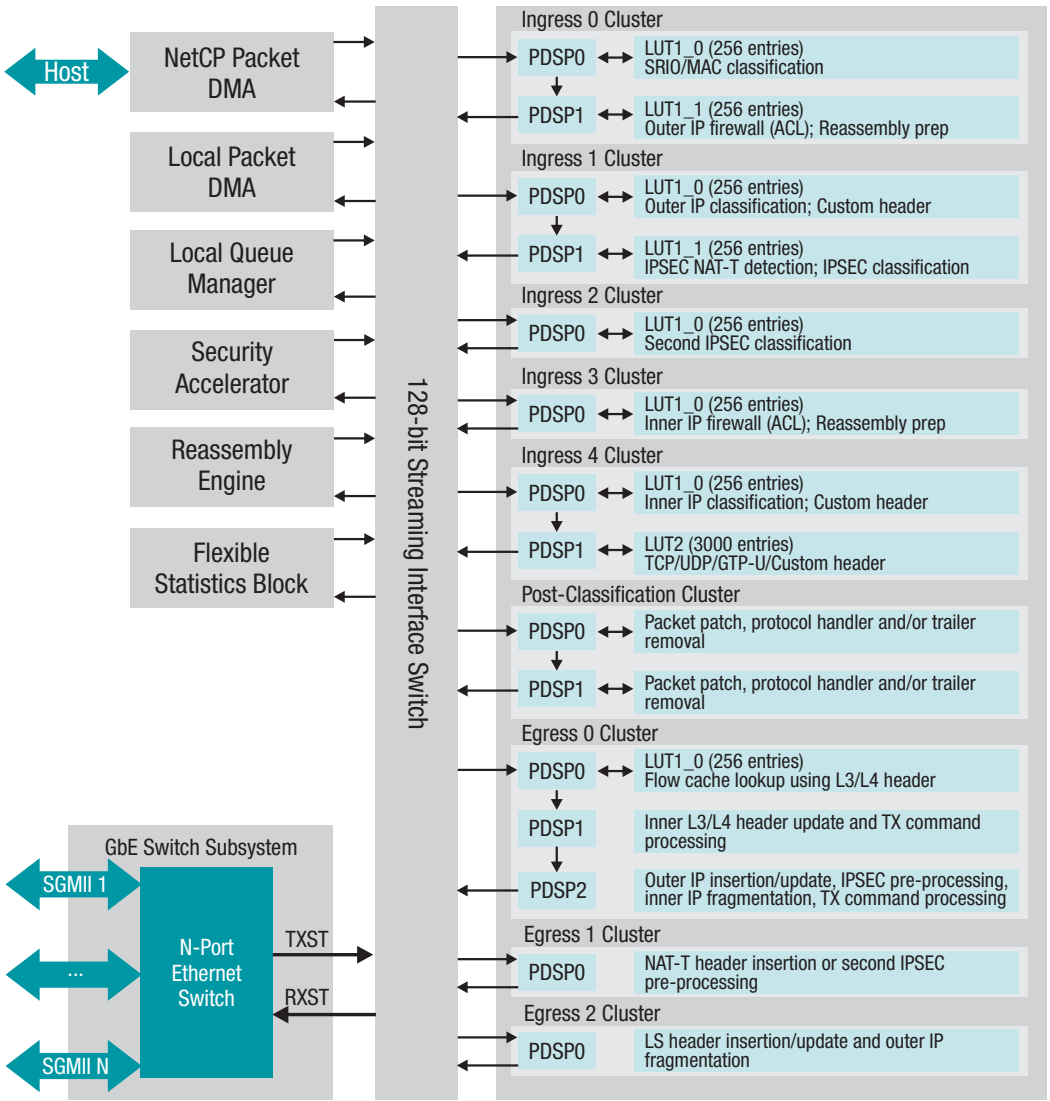

Figure 2. K2E Network Coprocessor packet accelerator

- Hardware-based packet arrival and departure time stamping to facilitate the implementation of network-based time synchronization such as IEEE 1588-2008. Time stamping of IPSEC-encapsulated 1558 is also supported.
- 802.1Qbb per priority flow control, per priority per port statistics, 802.1Qav egress shaping
- Host port is interfaced directly to packet processing subsystem (no hop to external memory required)

A second 3-port integrated switch is also included on the SoC, supporting two external Ethernet ports and one internal port, each running up to 10 Gbps. This switch also provides a MACSEC engine for line rate packet encryption and authentication.

Multicore Navigator: Hardware queuing and packet-aware DMA

K2E SoCs include a hardware-based queuing subsystem and packet-aware DMA engine that

is capable of interworking with packet/security accelerator hardware in the NetCP (discussed below), software and the hardware queues.

The hardware queuing subsystem consists of a hardware module (the queue manager) that exposes 8196 individual hardware queues via memory-mapped registers. Each queue can contain a list of 32-bit pointers to small chunks of memory called packet descriptors. Descriptors are allocated from blocks of memory that are initially registered with the queue manager. Descriptors are typically 64 or 128 Bytes in size and have a well-defined format that is understood by software, the accelerators and the DMA engine. Descriptors contain pointers to actual buffers that hold the packet data, but also have fields to hold various pieces of packet metadata such as actual buffer length, maximum buffer length and return queue number. Descriptors can be chained together to facilitate scatter/gather lists used by most operating systems.

The well-known queuing paradigms of **push** and **pop** are used throughout. Software or hardware accelerators can **push** a descriptor onto the tail (head) of a queue by writing the descriptor address (32 bits) to the appropriate memory-mapped register. The queue manager takes care of updating the previous/next links that manage the list. From ARM software, this **push** operation takes on the order of 10–15 cycles, assuming that the queue manager memory-mapped area is made "write buffer able" by the operating system. On the order of 90 additional cycles are typically required to convert from virtual to physical address and to issue the necessary memory barrier instructions required for ARM–DMA coherency to take place. Similarly, consumers can **pop** a descriptor off the top of the queue by reading from the memory-mapped address. From ARM software, this typically takes on the order of 400 cycles; including the 32-bit actual memory read, address conversion and cache line loading of the descriptor.

Certain queues are hardwired or can be assigned to the Packet DMA channels so that they can be used to send to/receive data from hardware accelerators. The DMA engine is aware of the descriptor format and can use fields out of the descriptor for specific purposes. Examples include:

- A queue may contain a list of descriptors to be used to place newly arrived packets (e.g., from the network or as output from an accelerator). The DMA engine will pop a descriptor from this assigned queue, parse it to obtain the buffer address where to place the data and the maximum size to transfer and then copy the new data into this memory.
- For queues used as inputs to accelerators, the DMA engine will again pop the top descriptor when the accelerator is ready for new data, parse it to obtain the buffer address and length, and then copy the data into the accelerator. The DMA engine will then read a return queue number from the descriptor and will push the [consumed] descriptor to this queue for recycling.

The DMA engine also supports more complicated descriptor and buffer allocation schemes. For example, a set of free queues, each with different sized buffers can be grouped together with a defined allocation policy. The DMA will select which free queue to use in the group based on the policy:

- Best fit: Select the queue whose buffer sizes match closest to the size of the incoming data
- Ordered fit: Use a buffer from the first free queue for the first N Bytes of the packet, and then use the second queue of the set for the next M Bytes, etc. This allows a packet to be

split into different memory areas and chained together.

Multicore Navigator: Quality of Service

The hardware queuing subsystem also contains several programmable microcontrollers that can be used to implement various functions, including packet scheduling, interrupt accumulation and egress traffic shaping to provide packet quality of service. The traffic shaper is implemented by firmware running on these microcontrollers. Each shaper consists of a set of hierarchical arrangements of hardware queues that hold packets that are fed from software or accelerators. The tip of the hierarchy feeds these packets to a configured output queue (this could be the input to the network coprocessor for example) using the configured shaping algorithm. The shaper firmware supports the following features:

- Packet drop scheduler, supporting configurable tail drop, head drop or random early drop (RED)
- Strict priority queues and best effort queues
- Weighted round robin (WRR) scheduling with configurable weights for each input
- Shaping/dropping can be done on Byte or packet counts

K2E Network Coprocessor packet accelerator

The packet processing accelerator (PA) as part of the K2E Network Coprocessor, consists of a collection of packet-processing microcontrollers known as PDSPs, coupled with look-up engines to perform pattern matching and classification of packet header fields. There are also dedicated hardware engines for in-line IPV4 and IPV6 fragment re-assembly and to perform a cyclic redundancy check (CRC) and checksum calculations. The K2E PA is connected to the rest of the SoC via hardware queues and the DMA engine discussed above.

PDSPs

PDSPs are small, 32-bit microcontrollers with an instruction set that has been optimized for packet header processing. Inside of the K2E PA, there is a broad-side interface that lets PDSP firmware see a sliding 32-Byte window view of each packet. Each 32-Byte packet chunk can be loaded into the PDSP registers in one cycle, as needed by firmware. This allows the PDSP firmware to work on packets in a streamlined fashion very efficiently. Each PDSP has access to its own program and data RAM.

Look-up engines

The K2E NetCP PA contains two types of look-up table (LUT) accelerators. The first-pass LUTs hold entries that are 64 Bytes wide. Each entry can contain up to 44 Byte-wide fields and 16 bit-wide fields. Various sizes of these LUTs are present, ranging from 64 entry tables to 512 entry tables. Each table is capable of look up at line rate. Entries can be added and deleted while packets are flowing through them. The first-pass LUT can also create an RSS hash of packet fields (typically IP source/destination addresses and ports).

There is also one second-pass LUT at the final ingress processing stage (see below). This contains up to 16K entries, each one 128 bits wide.

Processing flow

The PA is organized as a processing pipeline with processing stages roughly corresponding to typical packet protocol layering. A packet is received

into the PA directly from the switch subsystem, or from dedicated hardware queues. These hardware queues can be fed from software or from other accelerators as discussed above. While being processed by PA, a packet resides entirely in the K2E NetCP internal buffering. The PA pipeline can contain multiple packets simultaneously.

Ingress processing

On ingress, the accelerator consists of five clusters of PDSPs/look-up engines organized as follows:

- Ingress cluster 0: Layer 2 processing and outer firewall; Outer re-assembly preparation
- Ingress cluster 1: Layer 3 / first pass IP/IPSEC ESP/AH, NAT-T
- Ingress cluster 2: Second Pass IPSEC (bundled S) ESP/AH
- Ingress cluster 3: Layer 3/ inner IP processing and firewall
- Ingress cluster 4: Layer 4/5 processing including layer 4 checksums and L4 port look up

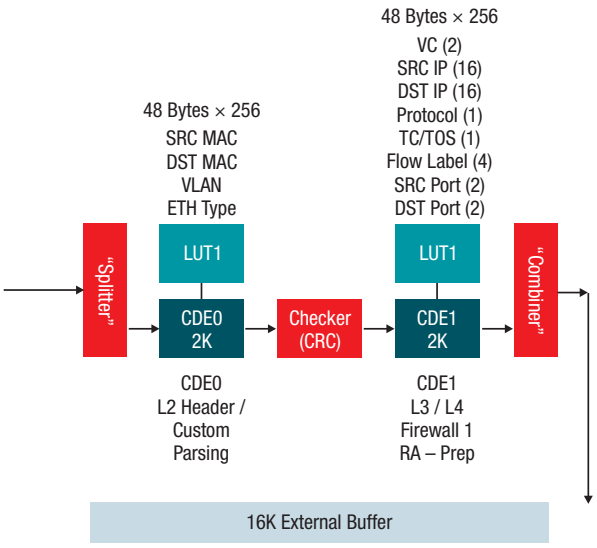

Figure 3. K2E Network Coprocessor ingress stage 0 architecture

At any stage, a PDSP cluster has the same basic hardware architecture. As an example, the first ingress stage cluster is illustrated in Figure 3 below. It consists of two look-up engines (labeled as LUT1 in the figure), coupled to two PDSPs (labeled CDEx) and a CRC/checksum hardware accelerator. Typically the first stage cluster PDSP firmware will perform Layer 2 classification and firewall checks on packets received from the switch. Roughly, the processing is as follows for this stage:

- Parse the L2 headers of the packet. Basic packet validity checks are done and certain exception packets may be identified here and passed immediately to software.
- Extract fields of interest such as MAC addresses, VLAN IDs, Ether-types
- Send these extracted fields to the first look-up engine. While waiting for the result, begin processing the next packet.
- The engine look up will return a match record from one of the 256 entries or failure result. On a match, the match record will contain instructions telling the firmware where to send the packet next (typically to the next stage of the pipeline). If the packet is to exit the PA (e.g., an exact match of a particular Ether-type says to send the packet immediately to the host), the match record will indicate where to send the packet (which hardware queue) and where to get a free buffer to place the packet in.

The second PDSP and look-up engine in the first ingress cluster can be used to perform a firewall function on the packet. Match results at this stage can:

- Drop the packet
- Pass the packet on
- Mark the packet. The mark can be used in subsequent stages.

The remaining ingress clusters have similar structure. A packet may be sent to the next cluster or exit the subsystem based on what the cluster firmware decides and the rules programmed into the look-up engines in that cluster.

On final egress of the subsystem, the packet will contain items of meta data that are useful to the software stack consumer:

- Offsets to the packet L2, L3, L4 headers are provided
- Markers defined by the application along with the look-up table rules will be present. These indicate the rule(s) that the packet matched along the pipeline. These can be used by the software stack to bypass its classification logic
- IP header and L4 checksum success/failure will be indicated
- The ingress Ethernet port is provided

Two hardware re-assembly engines are present to collect IP (V4 or V6) fragments of a flow and return completed IP datagrams to the pipelines. These effectively allow support for inner and outer fragments in one level of IP tunneling, particularly for targeting inline IPSEC applications.

Egress processing

The K2E NetCP PA contains three PDSP clusters for packet egress processing. These have similar structure to the ingress clusters and are designed for the following egress processing:

- Cluster 0: Implements a flow cache, can perform inner tunnel IP fragmentation, L4 checksum and preparation for IPSEC pass one
- Cluster 1: Implements outer IPSEC pre-processing (for second security transform in a security accelerator (SA) bundle)
- Cluster 2: Performs outer IP fragmentation and L2 framing.

The egress pipeline will perform framing and packet header/trailer construction on payloads for flows that are pre-programmed into the flow cache (at egress cluster 0). The flow cache consists of a look-up engine and memory that contains software-programmable packet framing/patching/header insertion instructions. The results from the ingress pipeline processing on a packet, for example, can be used as part of the look up. This allows no-host software touch packet forwarding with some level of header manipulation (e.g., for simple IP routing or NAT translation).

K2E Network Coprocessor security accelerator

The Security Accelerator (SA) provides multiple hardware engines for the following packet crypto operations:

- AES
- 3DES
- SHA1 HMAC
- SHA2 HMAC (up to sha256)
- Kasumi
- SNOW 3G
- ZUC

In addition, it supports various modes of the above algorithms such as HMAC, CBC and counter mode, as well as combined modes such as GCM and CCM. It allows encryption and authentication to be performed in one operation (e.g., AES CBC + SHA1). As it is hardware queue based, it can accept packets from software as well as other accelerators, in particular the PA. Finally, it contains PDSP engines to do packet security protocol processing, e.g., for IPSEC. This processing includes window

replay checks for ingress and sequence number maintenance for egress.

The SA is programmed by software with security contexts. These may be in external or internal memory, but are loaded by the SA when required into an internal context cache. These contexts hold keys for the encryption and authentication algorithms, as well as any state (for example the window replay bit mask). Software can lock contexts into the cache if necessary.

IPSEC inflow

The SA hardware and firmware, in conjunction with the PA, is designed to implement to perform inline IPSEC processing. This is a powerful feature that offloads much of ingress and egress IPSEC processing from the host operating system. On ingress, PA PDSP clusters/look-up engines can be programmed by software with the matching criteria for IPSEC tunnels. Typically this would include the source/destination IP address, the protocol type (e.g., ESP or AH) and the Security Policy Index (SPI) value, and a pointer to the corresponding security association. A packet matching this signature would then be sent by that cluster to the security accelerator, with the matched security context pointer included as part of the packet meta data. After IPSEC processing (including decryption, authentication, window replay checks) in the SA, the packet will be returned to the PA pipeline for subsequent processing; this might involve another trip to SA in the case of bundled security transformations (e.g., ESP+AH). Ultimately the decrypted and authenticated packet may land in a hardware queue serviced by the host application or network driver. With some modification (as discussed below), this packet can now be processed more efficiently by the host stack as major portions of the IPSEC processing layers can be skipped over.

Similarly, egress packets may bypass the IPSEC processing layers in the host stack and can be sent to NetCP with explicit instructions (passed as meta data) telling NetCP to perform the IPSEC processing and crypto transforms as the packet egresses the SoC (alternatively, the flow cache can be pre-programmed with these instructions).

Networking software

The K2E NetCP PA and SA provide for powerful hardware resources that can be applied to network workloads. But to use this hardware effectively requires a great deal of firmware and host software participation. Texas Instruments provides several layers of firmware, software and APIs to facilitate customer's usage of these resources.

PDSP firmware and low-level drivers

Firmware is provided as part of the K2E software development kits, typically bundled with the Linux Kernel. This firmware implements the packet processing functionality described above. For bare metal and Real-Time OS (RTOS) implementations on ARM, a low-level-driver software library, MCSDK, is included that provides APIs to configure the PA and SA accelerators, and to expose access to the multi-core navigator functions (e.g., queue push/pop).

Operating system integration

Integration of K2E networking features with a high-level operating system such as Linux requires a balance between exposing all hardware capability to software versus maintainability of the operating system network stack and drivers.

Texas Instruments has taken an approach that compromises between the two extremes. The K2E has exposed features such as NetCP's checksum offload, and traditional look-aside security acceleration to Linux as these have well-maintained interfaces to the kernel stack and drivers. Other examples in this category include exposing the hardware packet time-stamping capability of the integrated switch to socket-based user space applications via Linux's packet timestamp transport mechanism, and use of the RSS hash calculation that NetCP can perform to assign packet flows to processing cores.

TI has also integrated NetCP's inflow IPSEC feature into Linux by introducing a patch into the Linux kernel stack. This patch replaces the Linux native IPSEC Encapsulating Security Protocol layer (ESP) with a version that:

- Accepts provisioning of which security contexts can be offloaded
- On ingress, packets belonging to offloaded security contexts will be decrypted and authenticated "in flight" by PA and the security accelerator, prior to the packet reaching the kernel Ethernet driver. The replaced ESP protocol layer will check for such packets and will skip sending these to Linux's xfrm layer for crypto processing. Policy checking, and anti-replay window checking will still be done by Linux (i.e., PA/SA offload of these features will be disabled) as bypassing this software in Linux would require more intrusive changes to the Linux stack.
- On egress, the replaced ESP protocol layer checks for packets that have been assigned to offloaded contexts. For such packets, it skips sending the packet to the xfrm layer for crypto processing (either software crypto or look aside accelerator) and instead marks these with meta data. Ethernet driver checks for this IPSEC offload meta data in packets and when such packets are seen, sends them to the PA with instructions for the egress pipeline stages to perform the IPSEC processing and crypto transformation.
- Corner cases, such as the handling of fragmentation and re-assembly, require additional logic, primarily implemented in custom netfilter hooks that are installed by a TI-provided kernel module.
- A companion user space daemon is responsible for monitoring IPSEC security policy and association activity and issuing configuration commands to the PA and the security accelerator to offload/stop offload of the security associations.

What the NetCP means to the SoC

The features and functionality afforded by the NetCP, as used in conjunction with an embedded processor such as the K2E have traditionally been implemented with a standalone network interface chip (NIC). The NICs obviously add extra end-product-level bill of material costs, significant power consumption and board space. By including high-performance real-time packet and security processing as detailed, the K2E processor and its on-chip network coprocessor should be considered by every embedded developer with Ethernet network interface functionality as part of their end product application. The NetCP meets or exceeds many of the real-time network features and

performance requirements for today's embedded applications with its minimal power consumption compared to standalone network chips, affords end product efficiencies.

Acknowledgments

The author would like to thank his colleagues at Texas Instruments for their support in the preparation of this article.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265