High level programming model for heterogeneous MPSoCs using industry standard APIs

Peng Sun, Sunita Chandrasekaran and Barbara Chapman
Department of Computer Science, University of Houston

Abstract

Heterogeneous multicore embedded systems are everywhere. Programming these complex systems is becoming a daunting task. We need industry standards to solve multicore programming challenges. Multicore Association (MCA) offers an extensive set of APIs that support resource management, communication management and task management.

In our work, we use these APIs as a translation layer for a high-level programming model, OpenMP. This approach is to primarily abstract low-level hardware details from the programmer. The MCA communication library is one of popular APIs used in the embedded industry. As part of our on-going project, we worked on a Freescale QorIQ P4080 multicore platform that consists of eight state-of-the-art e500mc Power Architecture, integrated with Data Path Acceleration Accelerators (DPAA).

We extended the existing MCAPI implementation onto the Pattern Match Engine (PME) to provide required communication interface between Power cores and the DPAA PME. PME is a data path hardware accelerator designed to accelerate the regular expression scanning for data processing. While this hardware accelerator plays an important role in performance of the networking related computations, the programmers are still expected to know much of the lower level details on a set of domain specific APIs, which means the development on PME difficult and error prone.

In this work, we successfully mapped the MCAPI connectionless message onto the PME's Pattern Matching Control Interface library, abstracting the lower level configurations and function calls onto a relatively higher level industry standard libraries.

Introduction on MCAPI

MCAPI – MCA communication API

The purpose of MCAPI, which is a message-passing API, is to capture the basic elements of communication and synchronization that are required for closely distributed embedded systems. MCAPI provides a limited number of calls with sufficient communication functionalities while keeping it simple enough to allow efficient implementations. Additional functionality can be layered on top of the API set. The calls are exemplifying functionality and are not mapped to any particular existing implementation.

Figure 1. shows the three types of communication supported by MCAPI:

- Messages , connectionless diagrams
- Package Channel , connection oriented, unidirectional, FIFO package streams
- Scalar Channel , connection oriented, single word unidirectional, FIFO

Target Platforms

P4080DS Board

The P4080DS is a high-performance computing, evaluation, and development platform supporting the P4080 power architecture processor.

Figure 2 shows the preliminary block diagram of P4080. The P4080 processor is based upon the e500mc core built on Power Architecture and oering speeds at 1200-1500 MHz. It consists of a three-level cache hierarchy with 32KB of instruction and data cache per core, 128KB of untied backside L2 cache per core, as well as a 2 MB of shared front side cache. There are totally eight e500mc cores built in the P4080 processor. It also includes the accelerator blocks known as Data Path Accelerator Architecture (DPAA) that offload various tasks from the e500mc, including routine packet handling, security algorithm calculation and pattern matching. The P4080 processor could be used for combined control, data path and application layer processing.

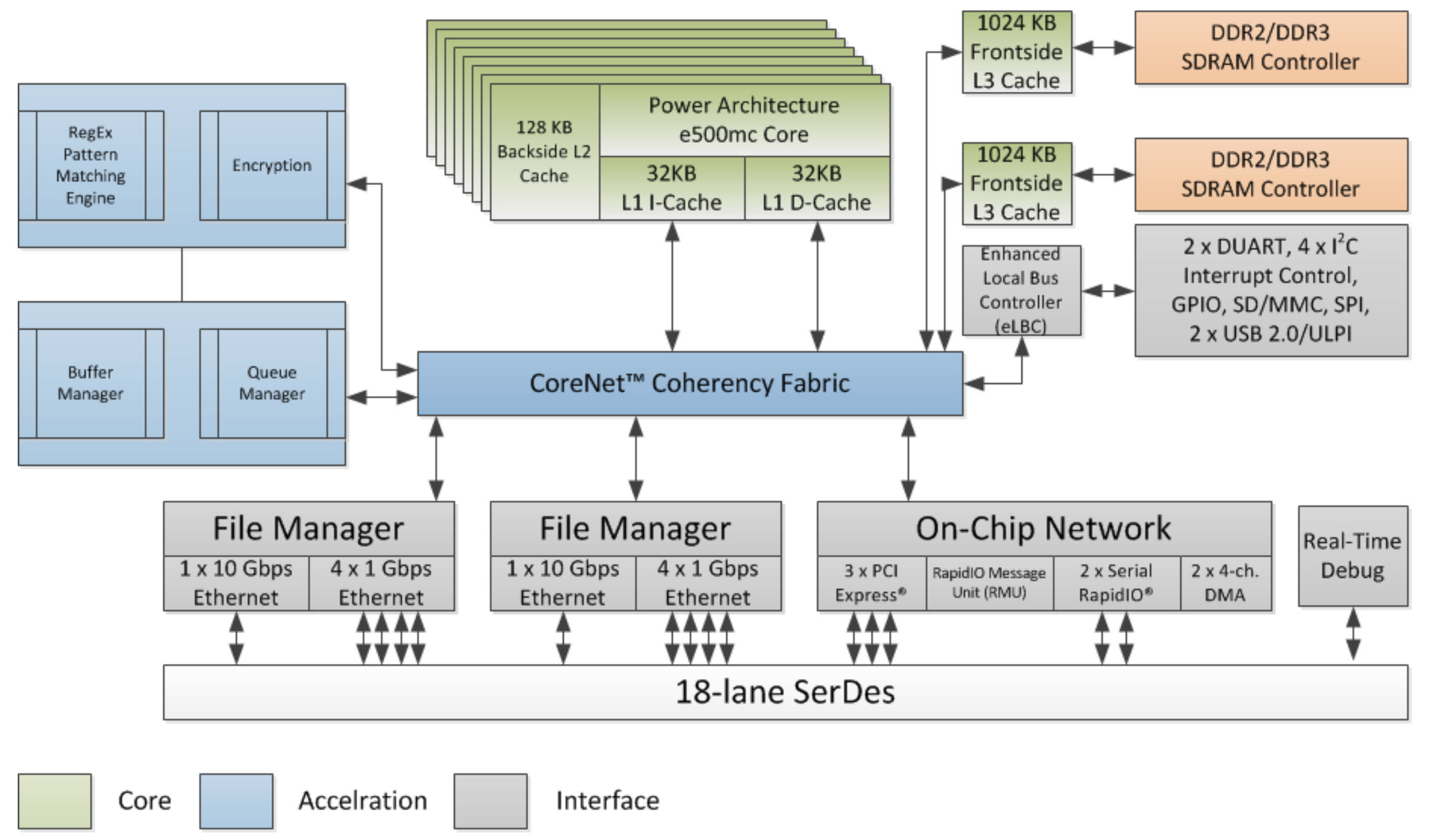
RegEx Pattern Matching Engine
Encryption
Buffer Manager
Queue Manager
128 KB Backside L2 Cache
Power Architecture e500mc Core
32KB L1 I-Cache
32KB L1 D-Cache
1024 KB Frontside L3 Cache
1024 KB Frontside L3 Cache
DDR2/DDR3 SDRAM Controller
DDR2/DDR3 SDRAM Controller
Enhanced Local Bus Controller (eLBC)
2 x DUART, 4 x I²C Interrupt Control, GPIO, SD/MMC, SPI, 2 x USB 2.0/ULPI
CoreNet™ Coherency Fabric
File Manager
1 x 10 Gbps Ethernet
4 x 1 Gbps Ethernet
File Manager
1 x 10 Gbps Ethernet
4 x 1 Gbps Ethernet
On-Chip Network
3 x PCI Express®
RapidIO Message Unit (RMU)
2 x Serial RapidIO®
2 x 4-ch. DMA
Real-Time Debug
18-lane SerDes
Core
Accelration
Interface

Figure2. P4080 Block Diagram

DPAA and PME

The P4080 includes the first implementation of the PowerQUICC Data Path Acceleration Architecture (DPAA). This architecture provides the infrastructure to support simplified sharing of networking interfaces and accelerators by multiple CPU cores.

The pattern matching engine is integrated into the DPAA system. The PME, showed in Figure 3, is our primary target for the MCAPI implementation. It is a self-contained hardware block capable of autonomously scanning data from streams for patterns that match a specification in a database dedicated to it.

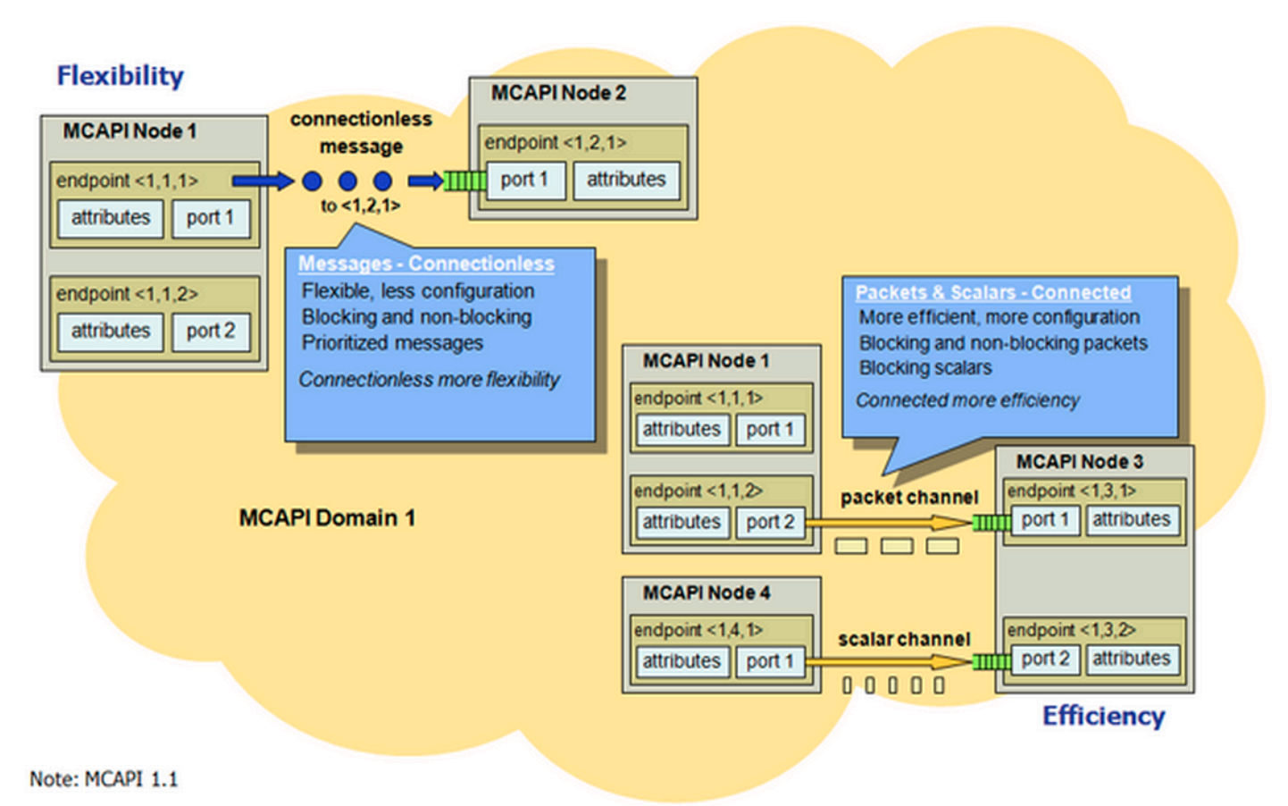

Figure1. The three kinds of communications supported by MCAPI,

Implementation of MCAPI on PME

We firstly studied functionalities and the prototype implementation of MCAPI. As mentioned earlier, MCAPI is an industry-standard API for inter-core communication within a loosely coupled distributed embedded SOC. It can be treated somehow like MPI, if lighter and designed for the embedded platforms. Nodes are a fundamental concept in the MCA interfaces that map to an independent thread of control, such as a process, thread, or processor. In our implementation phase, using P4080, we set up the PME as a "node" in MCA and e500mc processors as the other node.

We then explored the MCAPI transportation layer that will serve as the communication layer for PME. We modified the 'create end point' function to cater to the PME platform. We also modified the mcapi_trans_connect_ pktchan and mcapi_trans_open_pktchan_recv_i to build the MCAPI transportation channel for PME, that will appear on top of the DMA channel for PME package transportations.

For the MCAPI message passing, we came across the Pattern Matcher Control Interface (PMCI) library, a C interface to send and receive pattern matcher control commands to PME. We found that this functionality can be partly abstracted by utilizing the MCAPI massage mechanism, for e.g. mcapi msg send and mcapi msg recv. To implement the PMCI support within the MCAPI transportation layer, we added the PMCI shared library into the MCAPI build system. In this connectionless messaging implementation, the source code makes function calls directly to the PMCI library. The major functions involved in this process include:

- pmci_open
- pmci_close
- pmci_set option
- pmci_read
- pmci_write

For instance, the pmci_open has been mapped to the MCAPI transportation layer initialization subroutine. The pmci_close has been included in the MCAPI transportation finalize subroutine. The pmci_read and pmci_write have been mapped into the MCAPI message receive and send subroutines while the pmci_set option has been included into the MCAPI node and endpoint initializations.

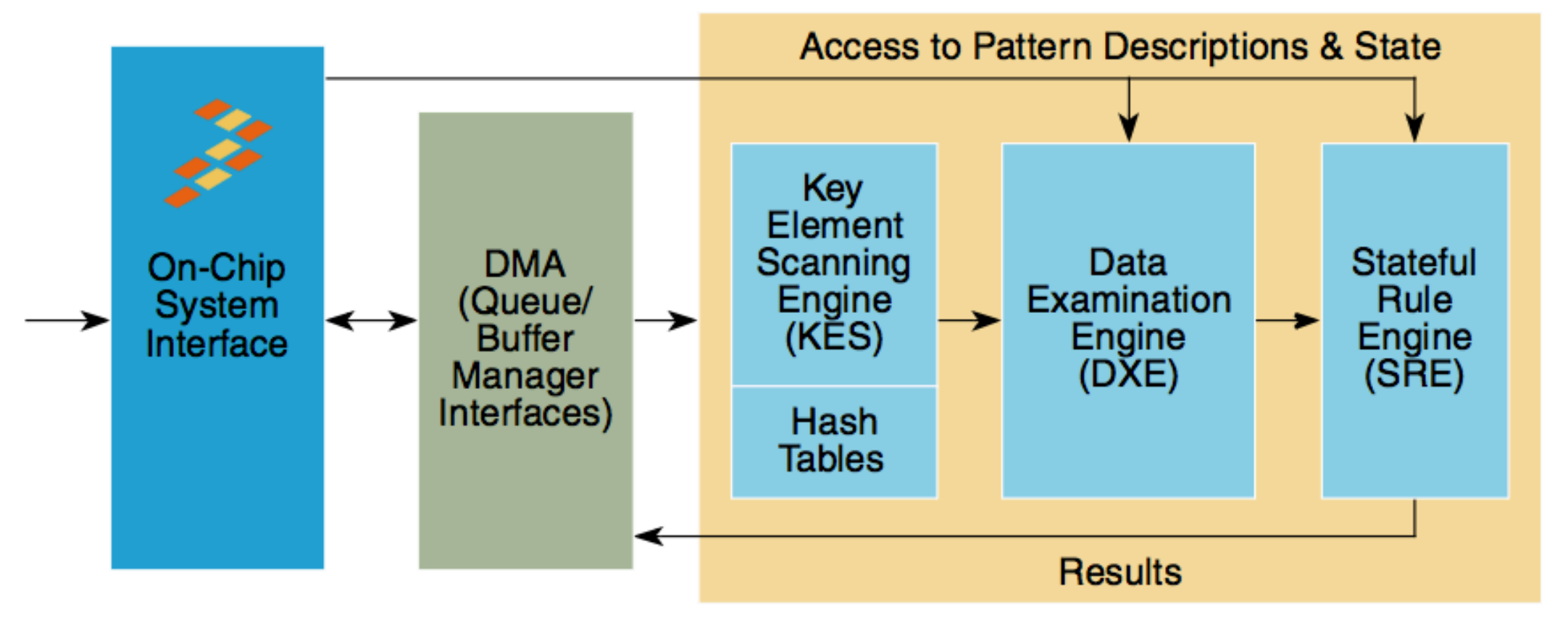
Access to Pattern Descriptions & State
On-Chip System Interface
DMA (Queue/ Buffer Manager Interfaces)
Key Element Scanning Engine (KES)
Hash Tables
Data Examination Engine (DXE)
Stateful Rule Engine (SRE)
Results

Figure 3. Pattern Matching Engine Diagram

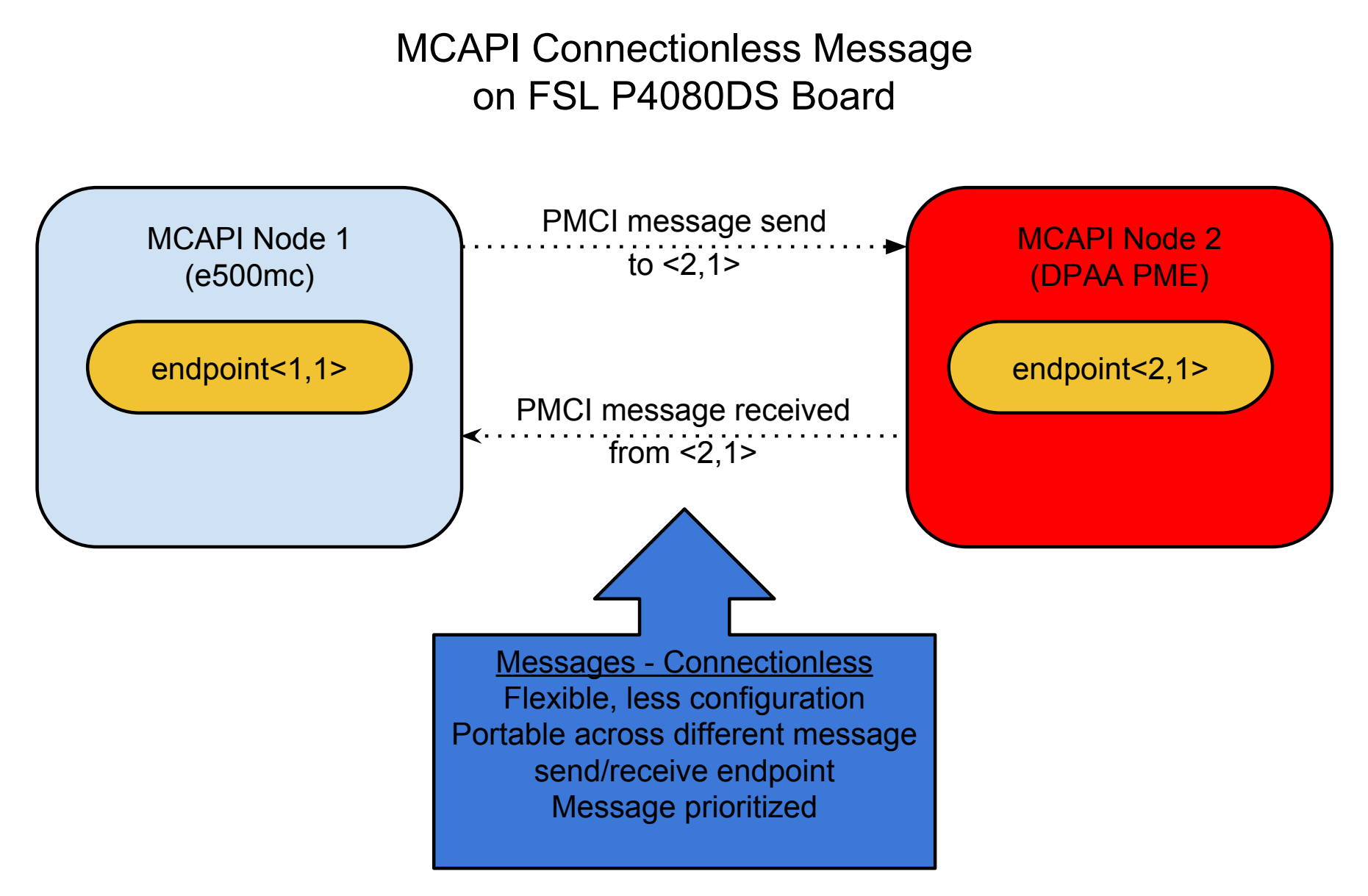
MCAPI Connectionless Message on FSL P4080DS Board
MCAPI Node 1 (e500mc)
endpoint<1,1>
PMCI message send to <2,1>
PMCI message received from <2,1>
MCAPI Node 2 (DPAA PME)
endpoint<2,1>
Messages - Connectionless
Flexible, less configuration
Portable across different message send/receive endpoint
Message prioritized

Figure4. MCAPI message on PME

Conclusions

In this poster, we have provided an overview of the MCA API, target platform and the approach adopted to create a portable programming stack for heterogeneous embedded platform. We are still in the learning phase and brainstorming ideas to use MCAPI to establish communication between the heterogeneous cores (in this case e500mc and PME). Our work in this poster serves as a preliminary and promising effort to abstract all the low level details of a hardware accelerator. We believe that MCAPI can serve as the right candidate to provide communication and synchronization mechanisms between underlying cores.

For the future work, we plan to create an even higher-level portable implementation by using MCA API as the target layer for OpenMP translation.

Acknowledgments

This research has been supported by grants from Freescale Semiconductor Inc. in association with Semiconductor Research Corporation (SRC)

UNIVERSITY of
HOUSTON
DEPARTMENT OF COMPUTER SCIENCE