



---

# Module 16 – System

---

**(ECE M116C- CS M151B) Computer Architecture Systems**

**Nader Sehatbakhsh**

**Department of Electrical and Computer Engineering**

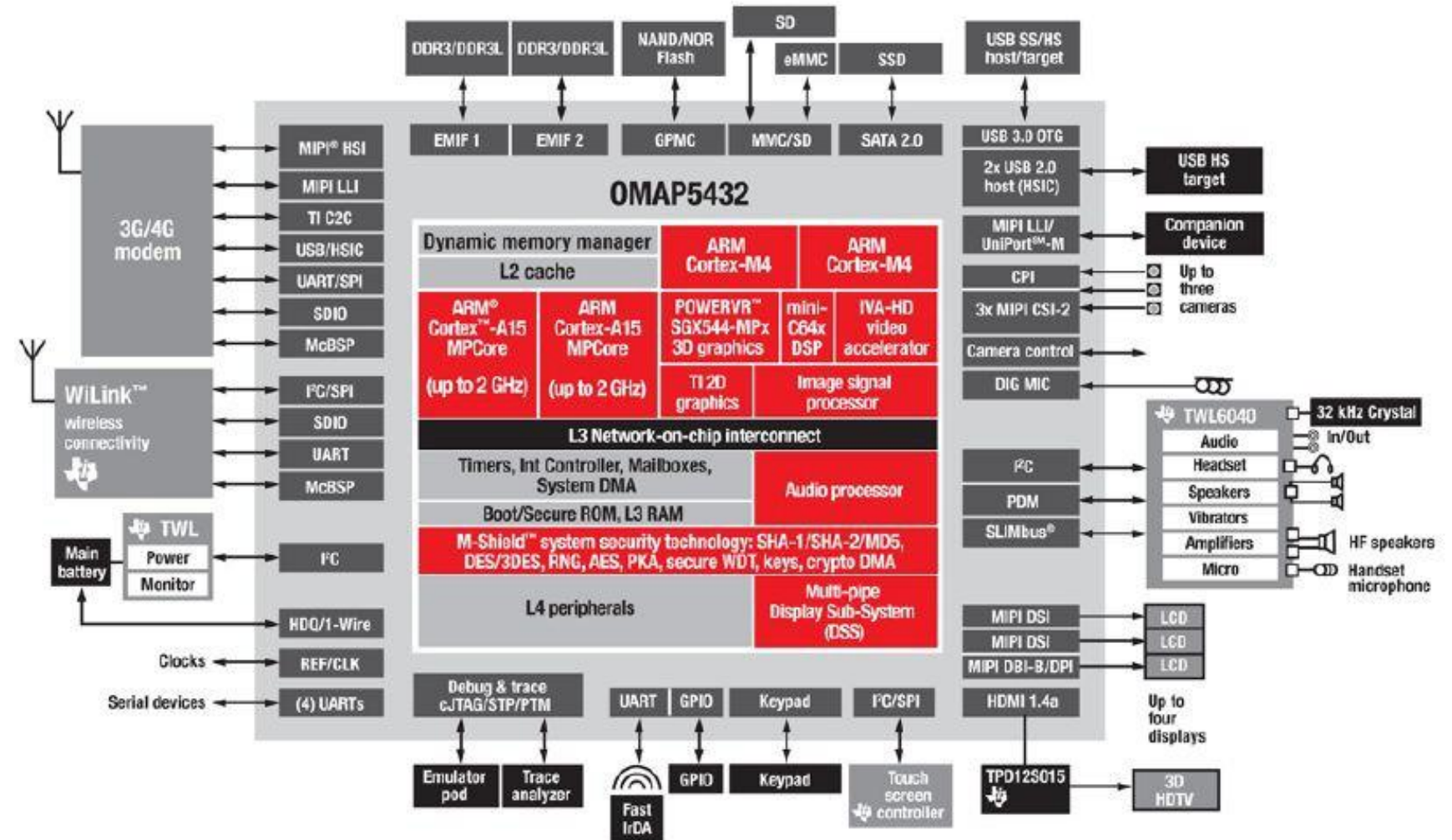
**University of California, Los Angeles**

# System

- There is more to a modern computer than just the processor and memory.
- Others components are generally referred to as Input/Outputs or I/Os.

# Example

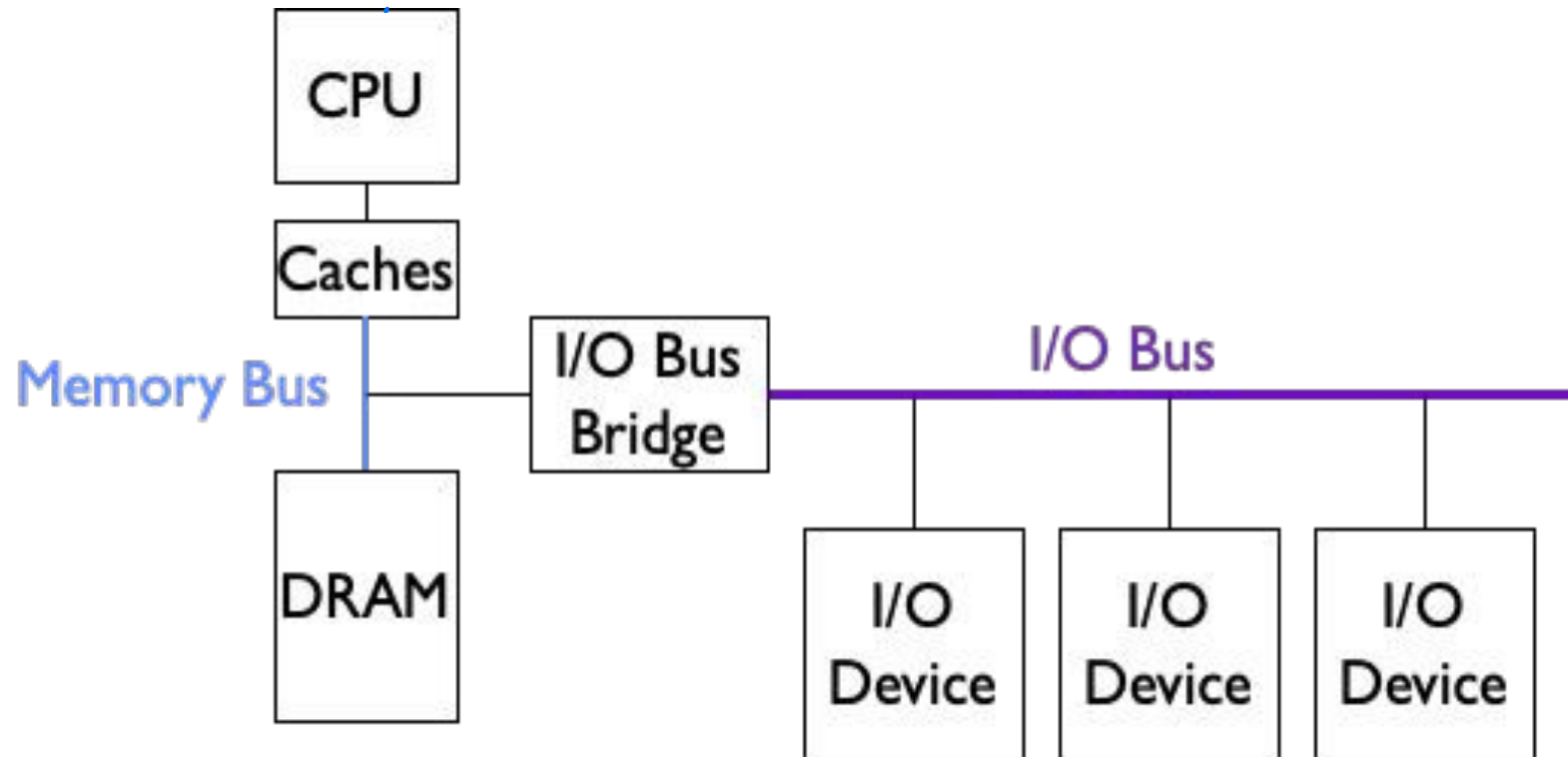
## TI OMAP5432 SoC



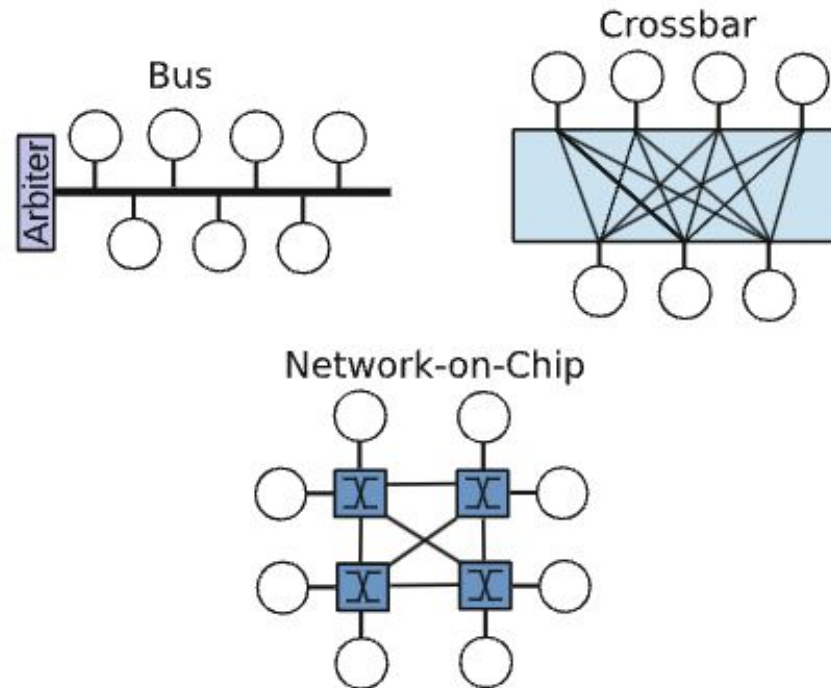
# I/O

- Various I/Os depending on the class of the computer (e.g., PC, Server, Mobile, Embedded)
- Broad categories:
  - Secondary/Tertiary storage (flash/disk/tape)
  - Network (Ethernet, WiFi, Bluetooth, LTE)
  - Human-machine interfaces (keyboard, mouse, touchscreen, graphics, audio, video, neural,...)
  - Printers (line, laser, inkjet, photo, 3D, ...)
  - Sensors (process control, GPS, heart rate, ...)
  - Actuators (valves, robots, car brakes, ...)

# Structure



# Other Options?



From: Network-on-Chip Design  
Haseeb Bokhari and Sri Parameswaran

# How to read/write from/to I/O

- Directly using Channels / Registers
  - Dedicate a specific set of registers to read/write (e.g., mainly in microcontrollers).
  - Having a dedicated channel (e.g., writing to a specific pin).

# How to read/write from/to I/O

- Memory-Mapped I/O

- Map a reserved part of the memory addresses to the I/O.
- Reading and writing to I/Os become loads and stores to specific addresses.
- Typically “uncached” memory addresses (using page tables).
- Virtual Address?



# How to communicate

- Different technologies and protocols (various needs and available technologies through decades)
  - Time multiplexed Shared and slow (very slow!) → PCI, IDE
  - Shared but fast(er) → Ethernet, SATA
  - Dedicated channel → PCIE
  - Handshaking *asynchronous* through arbiter, initiator, and target or fixed timing protocol.

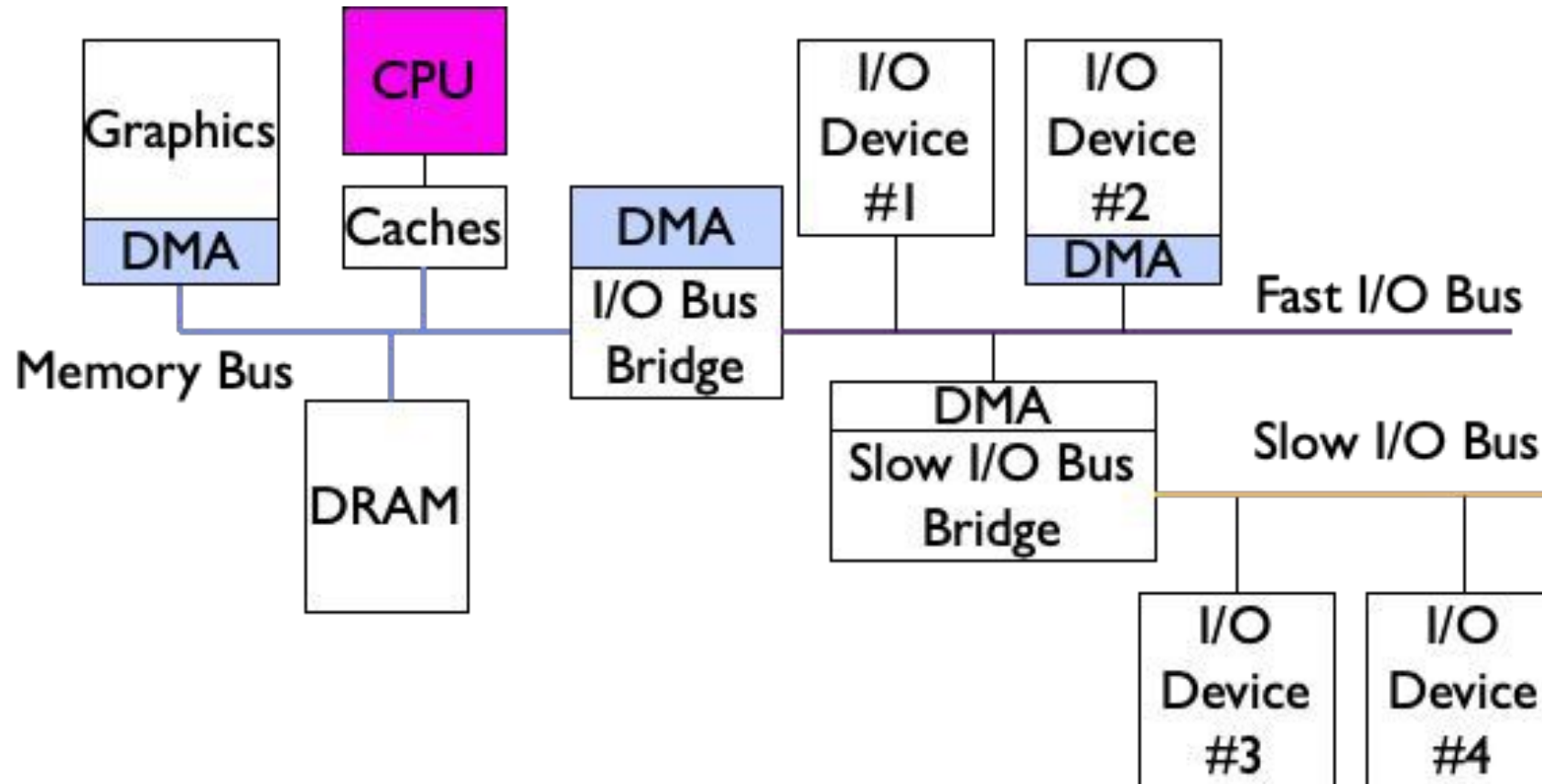
# DMA

- MMIO is slow, so why not talking directly to the memory?  
→ Use a separate unit called direct memory access (DMA)

# DMA

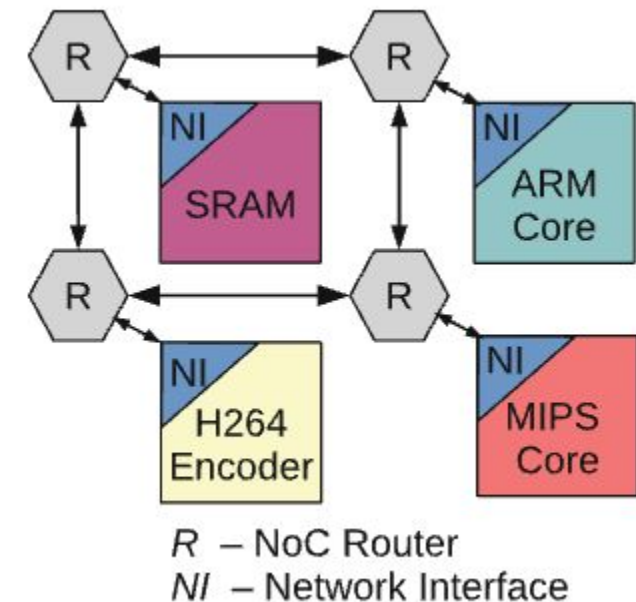
- DMA engines offload CPU by autonomously transferring data between I/O device and main memory. CPU can be notified when DMA is done with transaction.
- DMA programmed through memory-mapped registers.
- Some systems use dedicated processors inside DMA engines.
- Often, many separate DMA engines in modern systems.

# Modern I/O



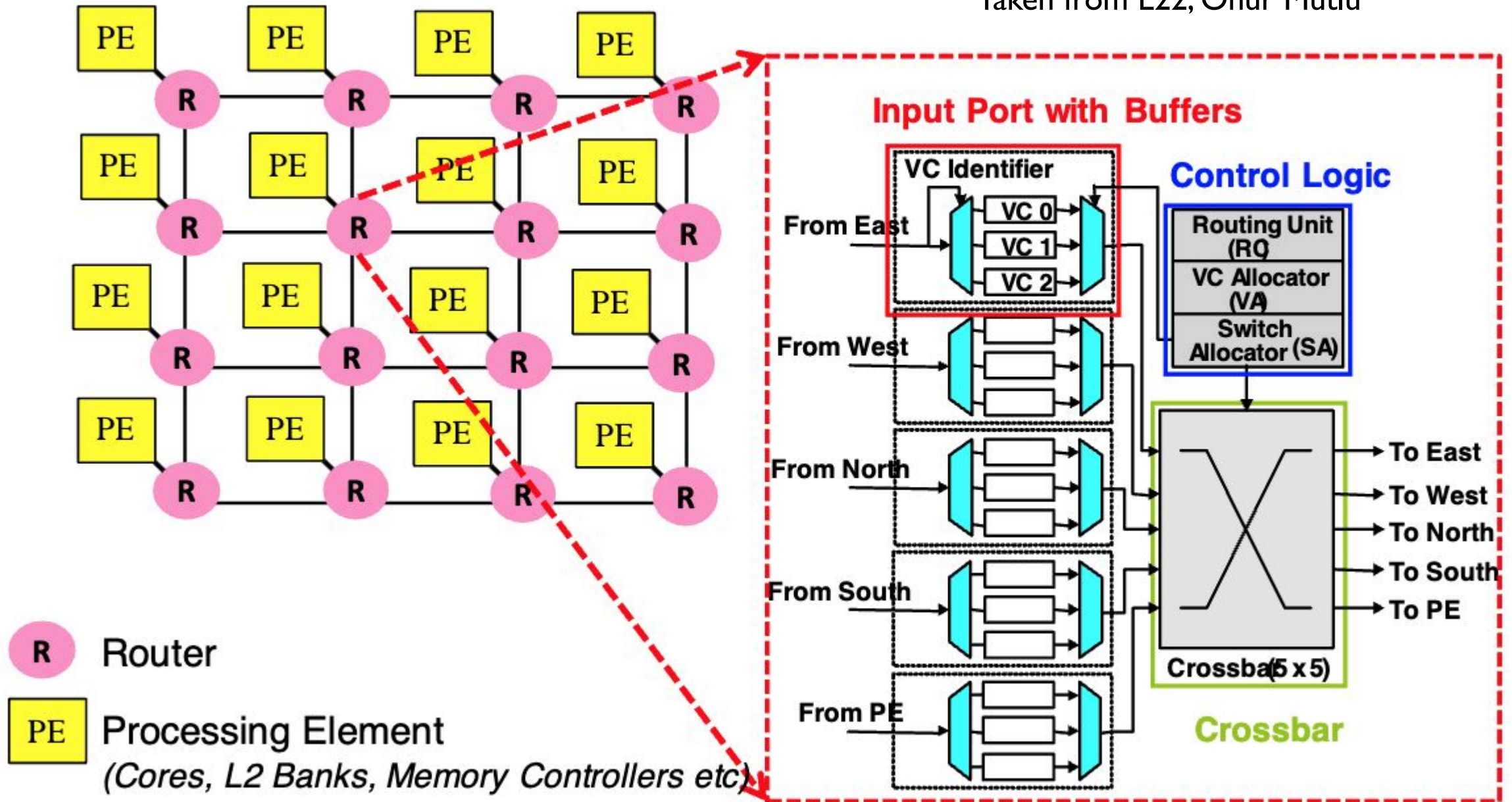
# Network on Chip (NoC)

- Different technologies and protocols
  - Multiple cores
  - Memory hierarchy (on-chip and off-chip)
  - GPU
  - Accelerators
  - I/O



From: Network-on-Chip Design  
Haseeb Bokhari and Sri Parameswaran

Taken from L22, Onur Mutlu



# Challenges

- Many challenges:
  - Performance optimization,
  - Scalability,
  - Energy efficiency,
  - Security,
  - Integration with emerging computing paradigms.
- Main design ingredients:
  - Topology, protocols, routing logic, router design, etc.
  - Bandwidth and latency → different technologies

# Interacting with Core

- Communication between I/O and CPU requires a timing mechanism.
- Interrupt vs. Polling.



# Interrupt

- If an event occurs, then call a service routine (a function stored in the reserved space of the main memory) to handle it.
- + no CPU overhead until the event happens
  - can happen at unexpected and unwanted times
  - large context-switch overhead

# Polling

- Checking the event with some predetermined frequency.
  - CPU overhead
  - Difficult to implement
  - + Can be fully scheduled

# Hybrid

- Interrupt on first event, use polling for some time, switch back to interrupt after sufficient time.

# Exception, Traps, and Interrupts



# Exception, Traps, and Interrupts

- Exception refers to an unusual condition occurring at run time associated with an instruction in the current thread/process.
- Trap refers to the synchronous transfer of control to a handler caused by an exceptional condition occurring within a thread (i.e., similar to jumping to a routine).

# Exception, Traps, and Interrupts

- Interrupt refers to an external event that occurs asynchronously to the current thread.
- To service an interrupt, an interrupt exception occurs, and the CPU subsequently experiences a trap.

# Exception, Traps, and Interrupts

- Example: *Branch mispredictions* causes an exception
  - To handle this exception, the CPU has to recover by flushing ROB, RS, etc.
  - A successful recovery requires maintaining all the necessary states for each instruction (e.g., RAT table), so we call this a precise exception.

# Exception, Traps, and Interrupts

- Example: *I/O events* causes interrupts.
  - To handle this interrupts, the CPU has to trap, and send the control to interrupt handler.
  - When to trap depends on the interrupts' priority.



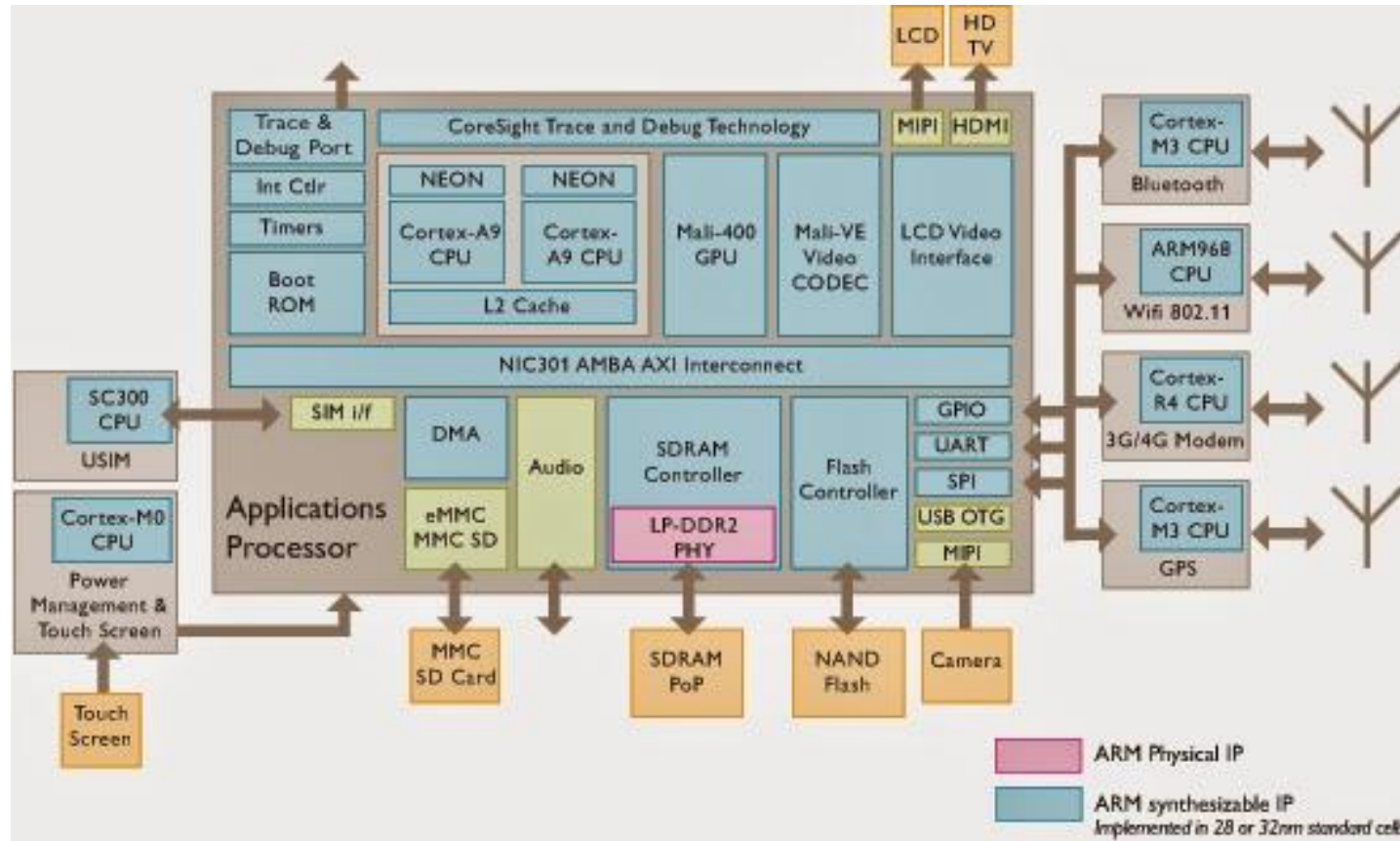
# Exception, Traps, and Interrupts

- Example: *OS System Call* causes a trap.
  - To invoke a service from the OS, the CPU has to execute a trap to transform the control into a handler.
  - Trap usually has a parameter indicating which service it needs. Handler uses a table to jump to the corresponding subroutine to handle the requested service.

# Exception, Traps, and Interrupts

- Other examples:
  - Unknown instruction → exception (issues a trap to handle)
  - Hardware failure → exception/trap
  - Overflow
  - ...

# System



# End of Presentation

# Acknowledgement

- This course is partly inspired by the following courses created by my colleagues:
  - CSI52, Krste Asanovic (UCB)
  - I8-447, James C. Hoe (CMU)
  - CSE141, Steven Swanson (UCSD)
  - CIS 501, Joe Devietti (Upenn)
  - CS4290, Tom Conte (Georgia Tech)
  - 252-0028-00L, Onur Mutlu (ETH)