Building Hardware Components for Memory Protection of Applications on a Tiny Processor

Oct 14 2017

Hyunyoung Oh*, Yongje Lee, Junmo Park, Myonghoon Yang and Yunheung Paek
Seoul National University, Korea

*Speaker

Outline

- Motivation
- Our Goal
- Overall Architecture
- Implementation Details
  - Security Interface
  - Memory Region Protector
  - Access Permission Matrix
- Experimental Results
  - Area Overhead
  - Performance Consideration
- Conclusion

Motivation

- In IoT era…
  - More and more small devices with Tiny processors
  - More sensitive user information
  - Memory protection is a conventional defense
  - Virtual memory cannot be applied due to high complexity

- Then How to Protect Memory?
  - MPU (memory protection unit in ARM) [3]
    - reconfigured in order to constrain different access permissions for every process
  - SMART [4]
    - is a new processor architecture including a special
  - TrustLite [7]
    - links code regions to data regions requires intrusive modification of an existing processor

Motivation

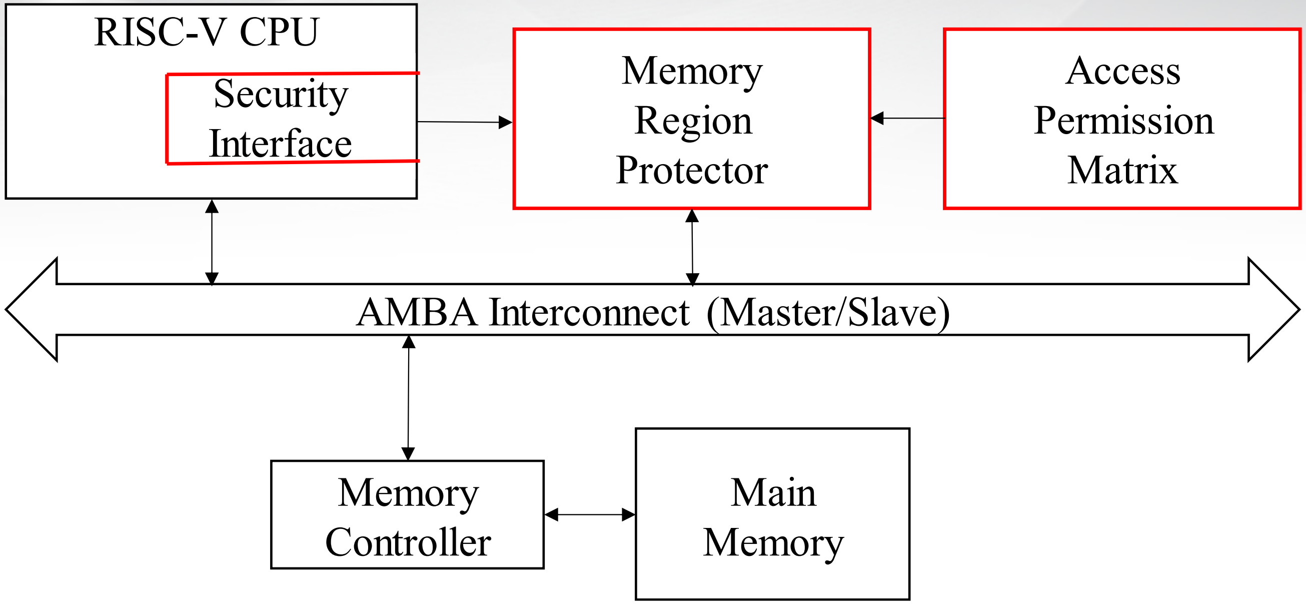
- In IoT era…
  - More and more small devices with Tiny processors
  - More sensitive user information
  - Memory protection is a conventional defense
  - Virtual memory cannot be applied due to high complexity

- Then How to Protect Memory?
  - MPU (memory protection unit in ARM) [3]
    - Inefficient
  - SMART [4], TrustLite [7]
    - Invasive and permanent modification of the existing host processor

Our Goal

- Secure and efficient memory protection mechanism
  - Minimize OS's role
  - Configure just once at the boot phase
- Less design change of the host processor
  - Conform to the modular design approach
  - Several hardware components can be assembled together

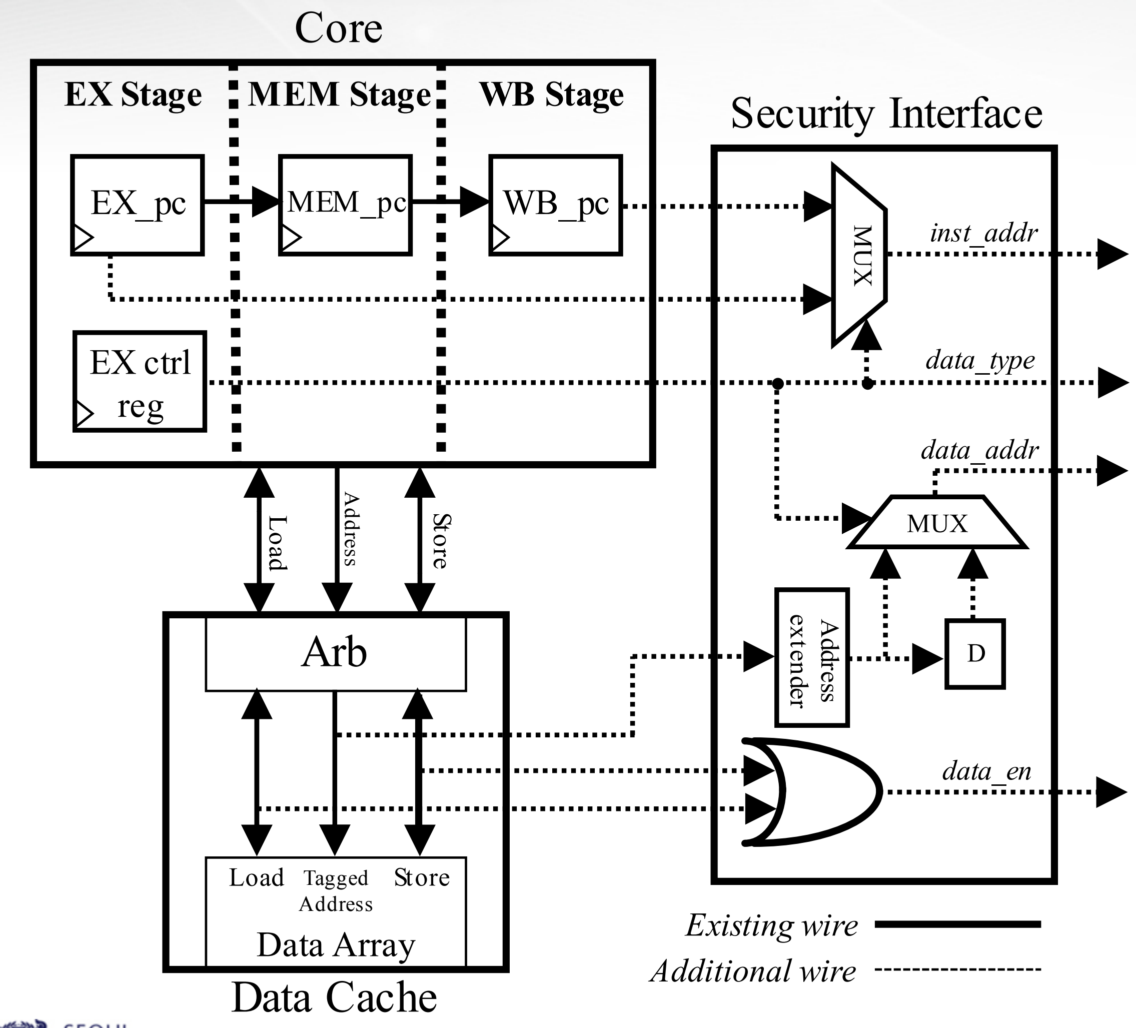
Overall Architecture



- 3 Main Hardware Components
  - Security Interface
  - Memory Region Protector
  - Access Permission Matrix

Implementation Details

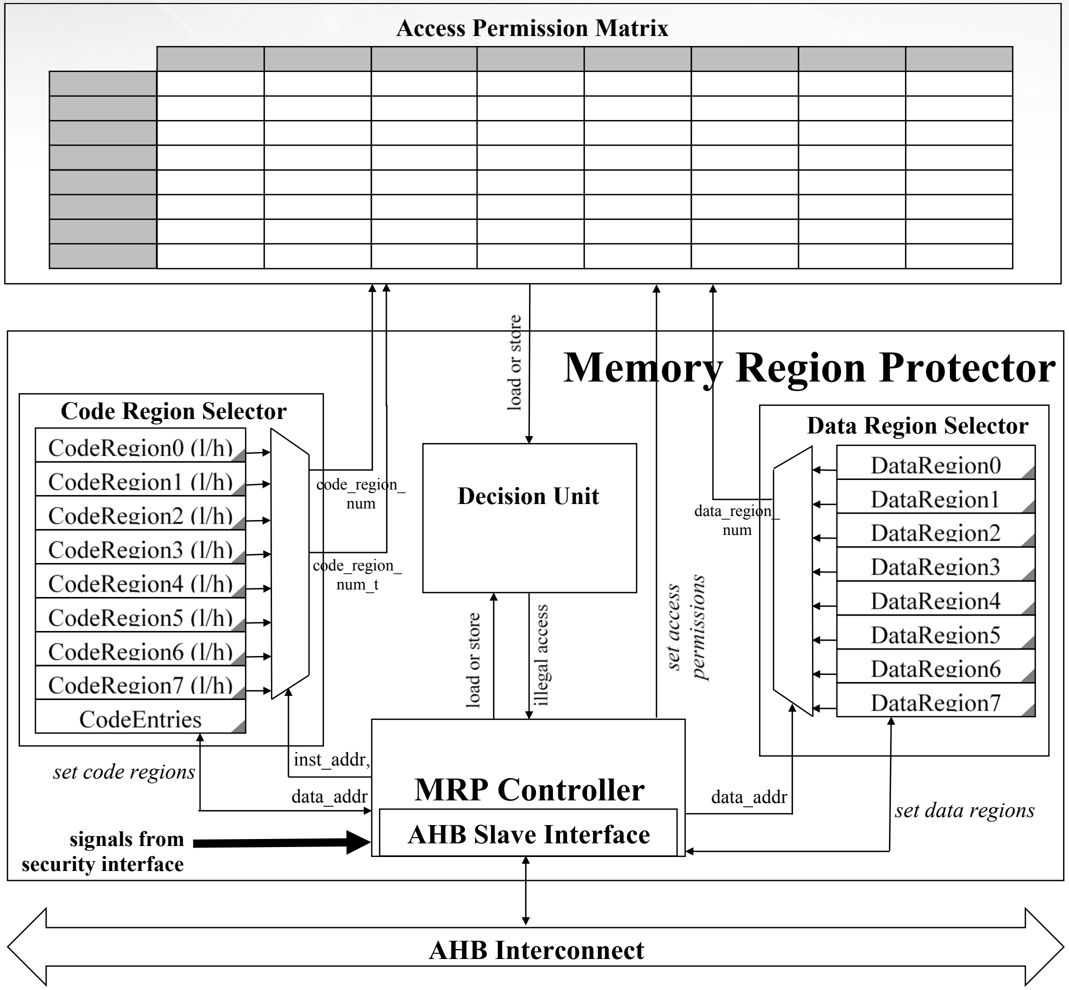
Security Interface



- Just connecting wires
- Extracting inst_addr, data_type, data_addr
- Synchronizing these 3 signals
- By referring EX control register

Implementation Details

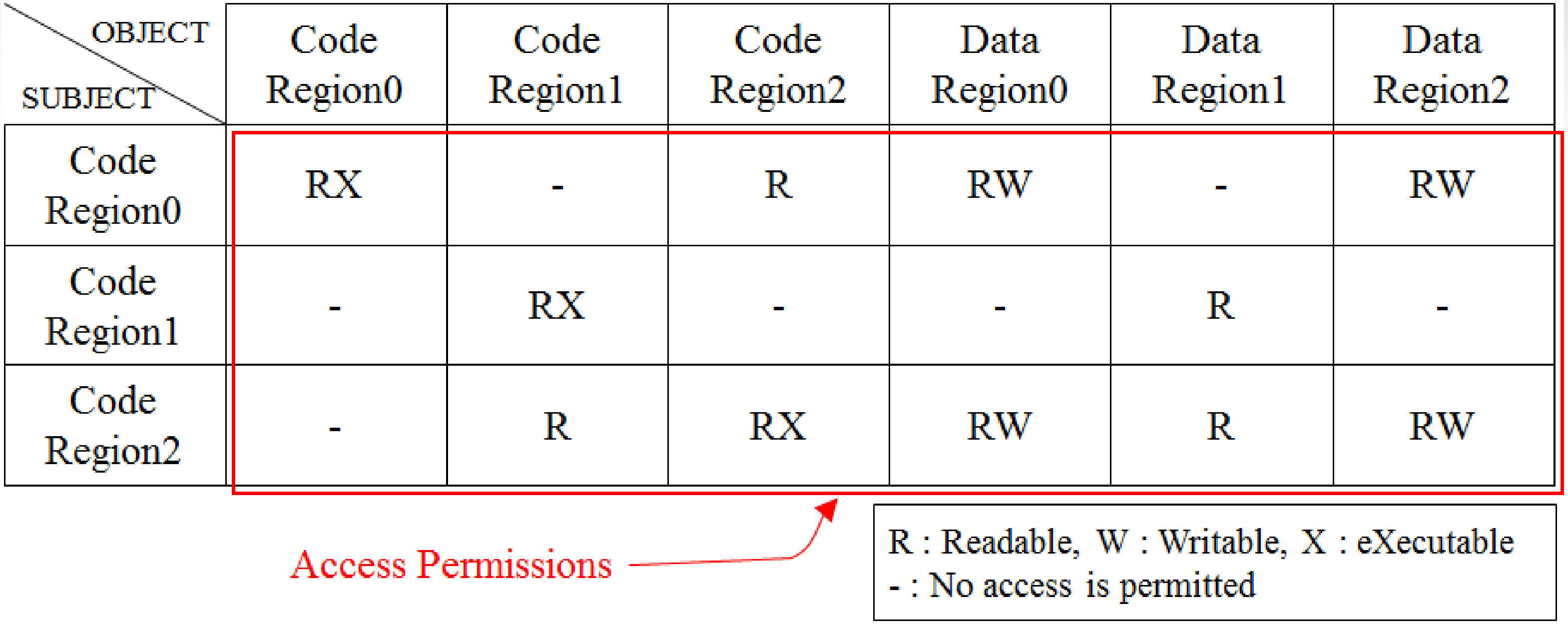
Memory Region Protector



- CRS/DRS classify the region indexes for the current instruction
- Access Permission Matrix provides the legitimate permission for those indexes
- Decision Unit checks whether the permission is violated or not

Implementation Details

Access Permission Matrix



OBJECT / SUBJECT	Code Region0	Code Region1	Code Region2	Data Region0	Data Region1	Data Region2
Code Region0	RX	-	R	RW	-	RW
Code Region1	-	RX	-	-	R	-
Code Region2	-	R	RX	RW	R	RW

Access Permissions

R : Readable, W : Writable, X : eXecutable
- : No access is permitted

- Has the access permission for code and data regions
- Check code-code access as well as code-data access
- Any access not permitted in the matrix will be illegal

Experimental Results

Area Overhead

- Xilinx Zynq-7000 board
- Version 1.7 of RISC-V Rocket core with DefaultFPGASmallConfig

Category	Components	LUTs	FFs
Baseline System	Rocket Core	9229	6894
Our Hardware Components	Security Interface	80	195
	Memory Region Protector	1066	1082
	Access Permission Matrix	36	204
	Total	**1182**	**1481**
	% over Baseline System	**12.81%**	**21.48%**

- 16.5% over baseline system in LUTs+FFs
- Memory Region Protector occupies 80% area within our total ← due to region boundary registers and selecting muxes

Performance Consideration

- Performance Overhead
  - Security Interface
    - Just probes wires so that incurs no impact to the critical path of the host CPU
    - Zero impact
  - Memory Region Protector
    - Runs in parallel with the functional execution of the host
    - Zero impact
  - Access Permission Matrix
    - In tiny processors, most applications are already fixed
    - Code/data region boundaries and their permission can be statically allocated
    - Negligible impact on the whole system performance

Conclusion

- Proposed Hardware Components
  - *Memory Region Protector* is the core component
  - This refers *Access Permission Matrix*
  - *Security Interface* extracts PC and memory target address
- Low Overheads
  - Low area overhead and near zero performance overhead
- More Flexible
  - In MPU [3] and PMP [5], region can be configured as a power-of-two multiple of 4KB
  - But we can set the boundaries by arbitrary addresses
  - Moreover, CPU internal information extracted through *Security Interface* can be used for various hardware based security mechanisms

Q&A

Thank You

Hyunyoung Oh (hyoh@sor.snu.ac.kr)
- 2007~2017: RTL Engineer in Samsung Electronics
- 2017~ : Pursuing PhD in Seoul National University
  Prof. Yunheung Paek is supervisor