Towards Secure Cyber-Physical Systems for Autonomous Vehicles

Morteza Biglari-Abhari

Department of Electrical, Computer & Software Engineering

University of Auckland

Email: m.abhari@auckland.ac.nz

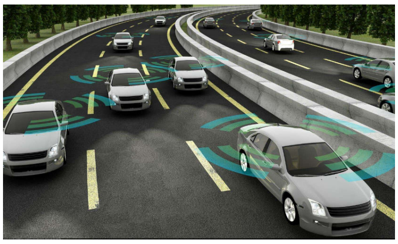
5 March 2020 – Barcelona Supercomputing Centre

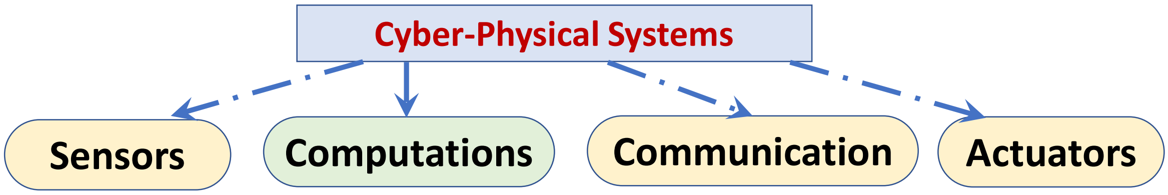
Rapid Market Growth: *(based on Allied Market Research estimates)*

The autonomous vehicle market will grow from *$54.23 billion* in 2019 to *$556.67 billion* in 2026.

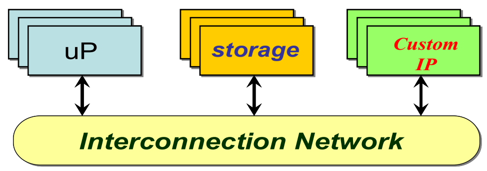
Data Security & Privacy Concerns Growth:

- **Remotely hacking modern cars**
  - Jeep digital systems hacked remotely to control the brakes and steering wheels [2014]
  - hackers tricked Tesla's Autopilot into suddenly changing lanes [2019]
- **Distributed Denial of Service (DDoS) attacks**
  - Mirai malware [2016]: creates botnet to launch Distributed Denial of Service (DDoS) attacks
  - Another version of it [Jan. 2018] targeted ARC processors based devices running Linux





- MPSoC architecture is optimized ***based on the application characteristics, design constraints*** (***performance, power/energy consumption, time predictability***), ***cost*** and ***security requirements***.



Each *uP* may have ICache and DCache (or multi-level caches).

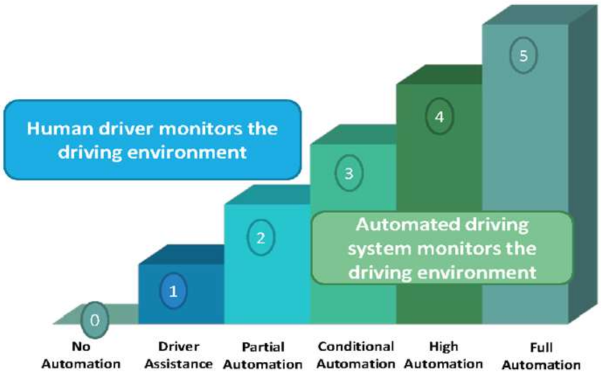
Domain-Specific Hardware/Software Co-Design methodology can be used to address the required design constraints and ***reduce design and verification time***.

Outline

- **Computational Requirements (from ADAS to ADS)**
  - **Pedestrian Detection**
  - **Adaptive Vehicle Detection**
- **Mitigating Security Vulnerabilities**
  - **Security Vulnerabilities and Threats**
  - **Security-Aware System Design Flow**
- **Some Related Works**
- **Summary and Conclusions**

Autonomous Driving Systems (ADS) Requirements

- **High Level of Accuracy** *(i.e. robust and reliable object detection for different environment conditions)*
- **Hard Real-Time Guarantees**
- **Emphasis on Very High Level of Safety and Reliability**
- **Emphasis on Very High Level of Security**
- **Addressing other marketing issues** *(cost reduction, less energy consumption, reducing CO_2 emission …)*



Source: Society of Automotive Engineers (SAE)

Cars with level 3 autonomy

- Cadillac CT6, Mercedes Benz E Class, Volvo S90

ADAS Typical Tasks:

- **Pedestrian Detection**
- **Vehicle Detection**
- **Adaptive Cruise Control**
- **Lane Departure Detection**
- **Traffic Sign Detection**
- **Parking Assistance**

Active Sensors: **The sensor emits a signal and then measures its reflection.**

- LIDAR, SONAR

Microsoft Kinect uses an IR transmitter and an IR camera.

Passive Sensors: **The sensor detects the radiation that is emitted, reflected or scattered by the object.**

- **Camera** is the most commonly used passive sensor.

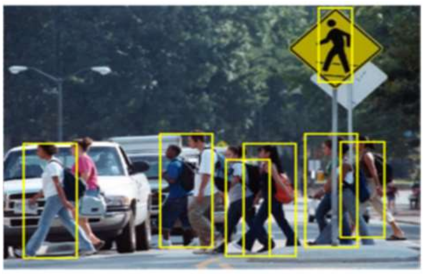
- ***Active sensors*** **are usually very expensive. (for example, LIDAR in Google autonomous car costs about $75000), while** ***passive sensors*** **(i.e. Camera) are cheaper and more environment friendly.**

Pedestrian Detection

Pedestrian detection is considered as one of the most challenging tasks in several domains such as surveillance, robotics, and driver assistance systems, autonomous driving systems, …

- ➢ Due to the variation of appearance and human poses

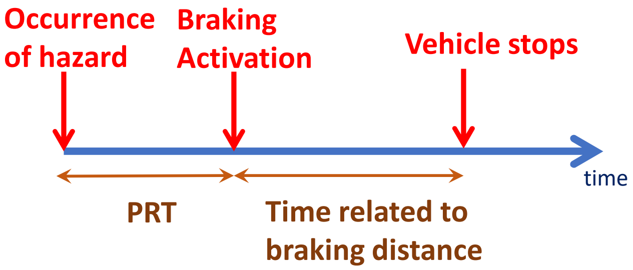
Source: www.eurocarnews.com



Source: www.pedestrian-detection.com

High computational complexity of real-time pedestrian detection:

- ➢ PRT (Perception-brake Reaction Time): between ***0.7 S to 1.5 S***
- ➢ Braking distance: ***14.84 m*** (50 km/h) to ***29.16 m*** (70 km/h) considering 6.5 m/S^2 deceleration



Occurrence of hazard

Braking Activation

Vehicle stops

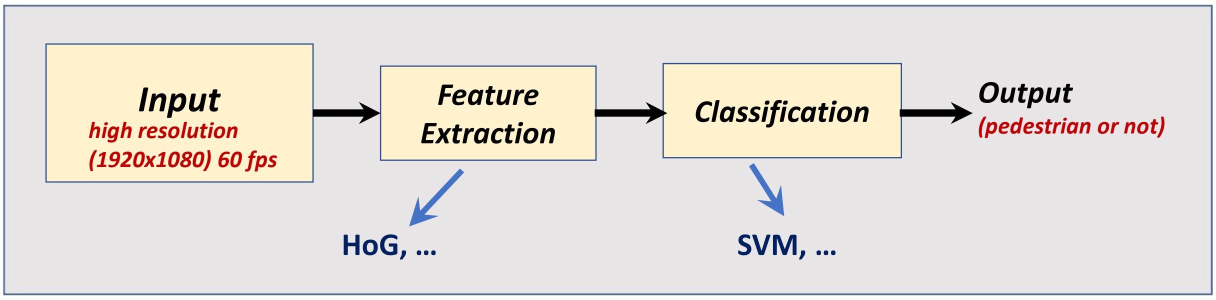
time

PRT

Time related to braking distance

- ➢ **Stopping distance** = perception-reaction distance + braking distance = ***35.68 m*** (50 km/h) to ***58.23 m*** (70 km/h) assuming PRT is 1.5 S

- ***Consequently less than a second time is left for all the processing required.***



Input
high resolution (1920x1080) 60 fps

Feature Extraction

Classification

Output
(pedestrian or not)

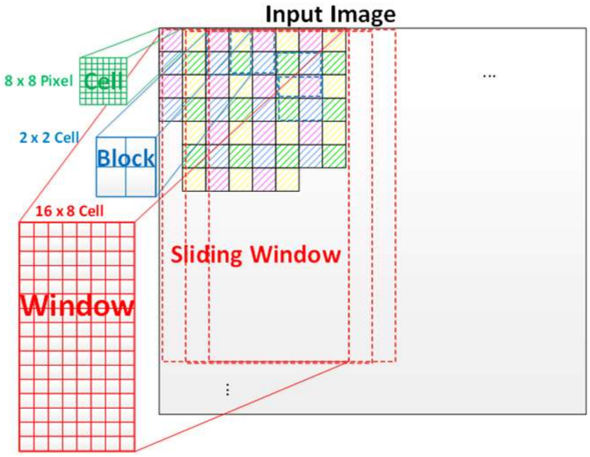
HoG, …

SVM, …

The basic idea for using **Histogram of Oriented Gradients (HoG)** is that ***the local appearance of an object can be described by its local intensity gradient distribution***.

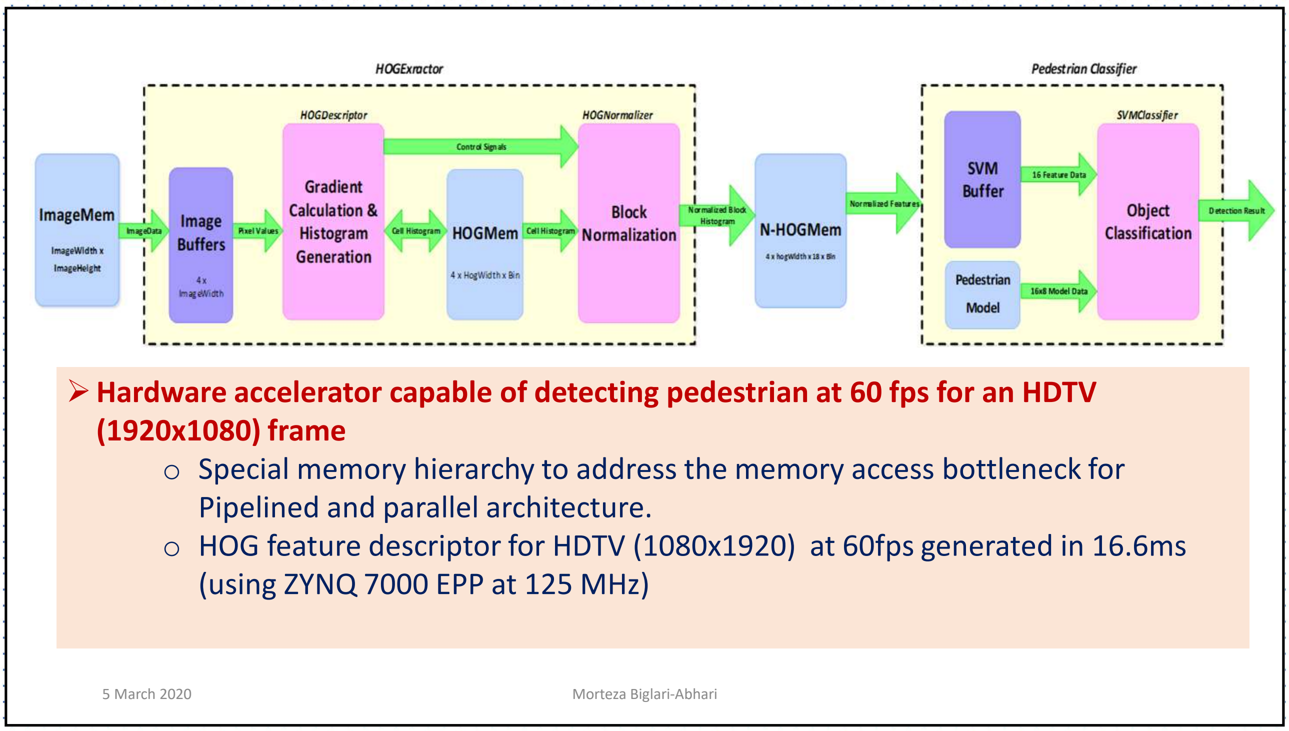
Histogram of Oriented Gradients (HOG)

- ➢ Gradients are computed within the cell
- ➢ Gradients generate an orientation histogram
- ➢ Generated histograms are normalized within the blocks to suppress the effect of local brightness and contrast

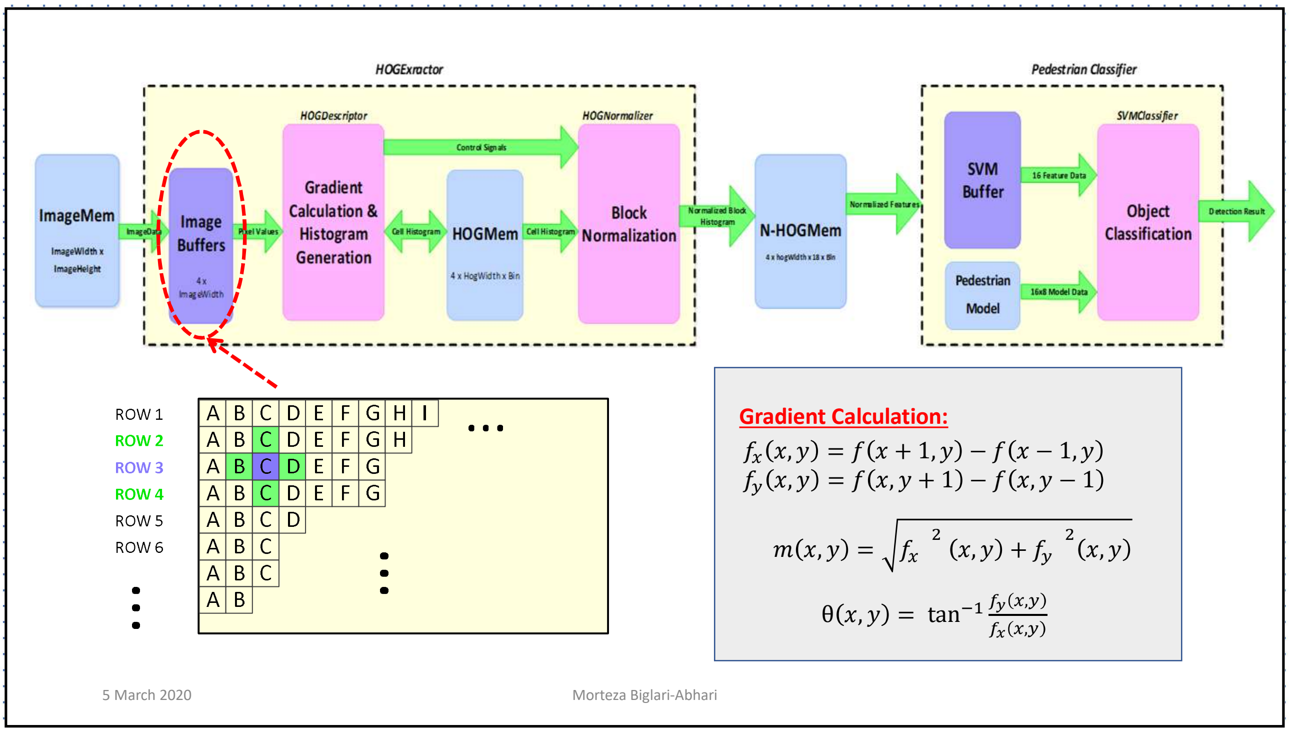


Support Vector Machin (SVM)

- ➢ Sliding window of 128x64 pixels
- ➢ Features are compared with the trained model through their dot product



- **Hardware accelerator capable of detecting pedestrian at 60 fps for an HDTV (1920x1080) frame**
  - Special memory hierarchy to address the memory access bottleneck for Pipelined and parallel architecture.
  - HOG feature descriptor for HDTV (1080x1920) at 60fps generated in 16.6ms (using ZYNQ 7000 EPP at 125 MHz)



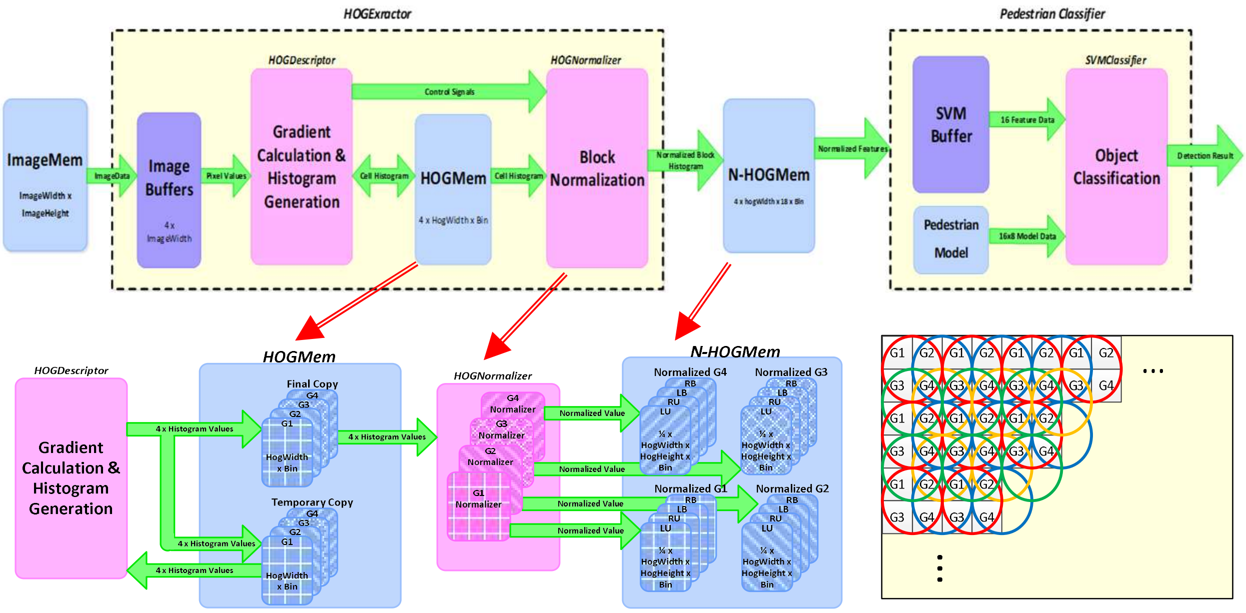
ROW 1	A	B	C	D	E	F	G	H	I	…
ROW 2	A	B	C	D	E	F	G	H		
ROW 3	A	B	C	D	E	F	G			
ROW 4	A	B	C	D	E	F	G			
ROW 5	A	B	C	D						
ROW 6	A	B	C							
	A	B	C							
	A	B								
⋮				⋮						

Gradient Calculation:

$$f_x(x,y) = f(x+1,y) - f(x-1,y)$$
$$f_y(x,y) = f(x,y+1) - f(x,y-1)$$

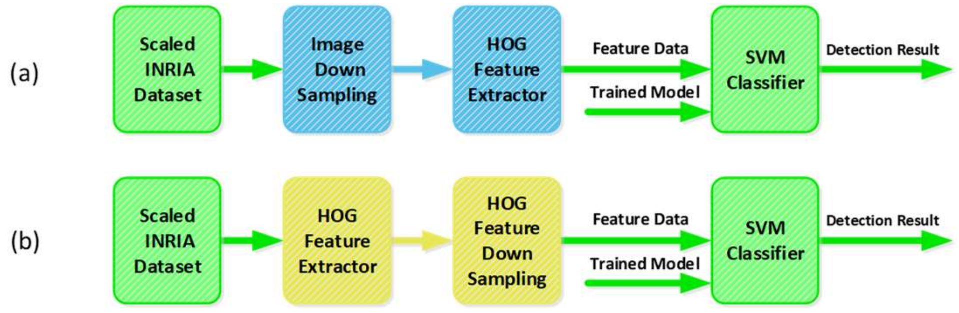
$$m(x,y) = \sqrt{f_x{}^2(x,y) + f_y{}^2(x,y)}$$

$$\theta(x,y) = \tan^{-1}\frac{f_y(x,y)}{f_x(x,y)}$$



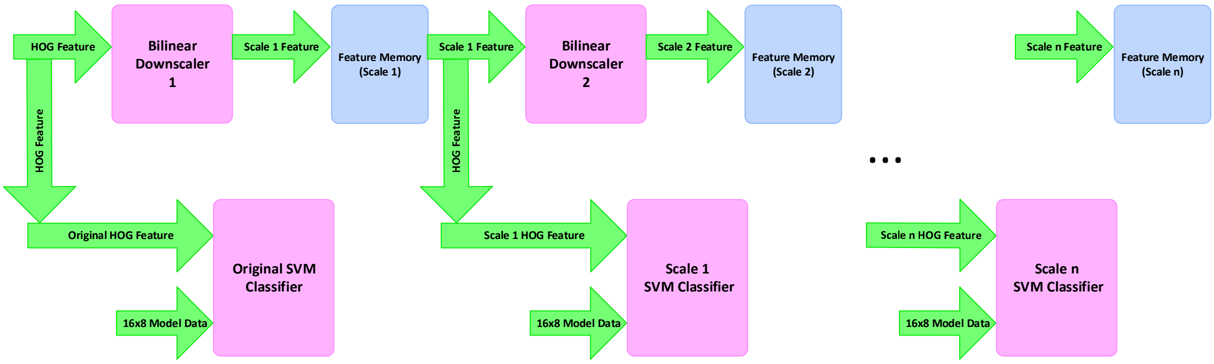
Multi-Scale Pedestrian Detection:

Multi-scale detection is required to cover different object sizes and distances to the vehicle.



To reduce the computational complexity, in our new approach the normalized HOG features are down-sampled to detect different sizes of the pedestrian.

- ***Our modified approach outperforms for the scales up to 1.4***
- ***Real-time multi-scale pedestrian detection is achieved for an HDTV (1920x1080) at the rate of 60 fps.***



Vehicle Detection

Vehicle detection approaches are considered for different environmental conditions:

- Detection in ***day***
- Detection at ***night***

Daytime detection methods focus on:

- The visual appearance of the vehicle
- Features such as symmetry and shadow under the car

Most of the detection during night time relies on the information of taillights.

Vehicle itself is not a static object and its appearance may change in different lighting conditions.

Robust detection requires using the features that:

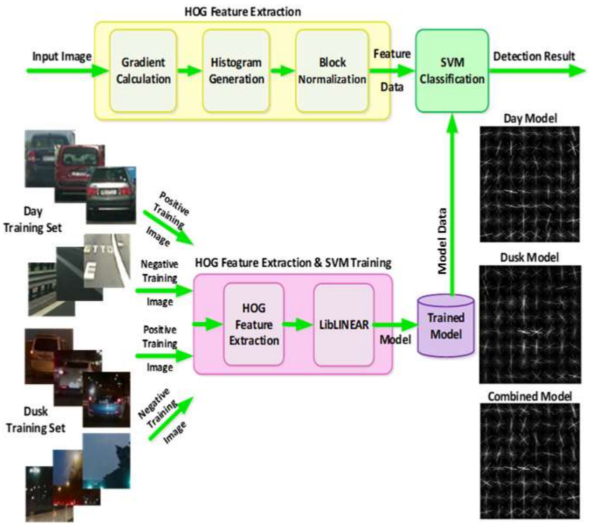
- Minimize the lighting and luminance variance
- are less affected by the change of environmental conditions

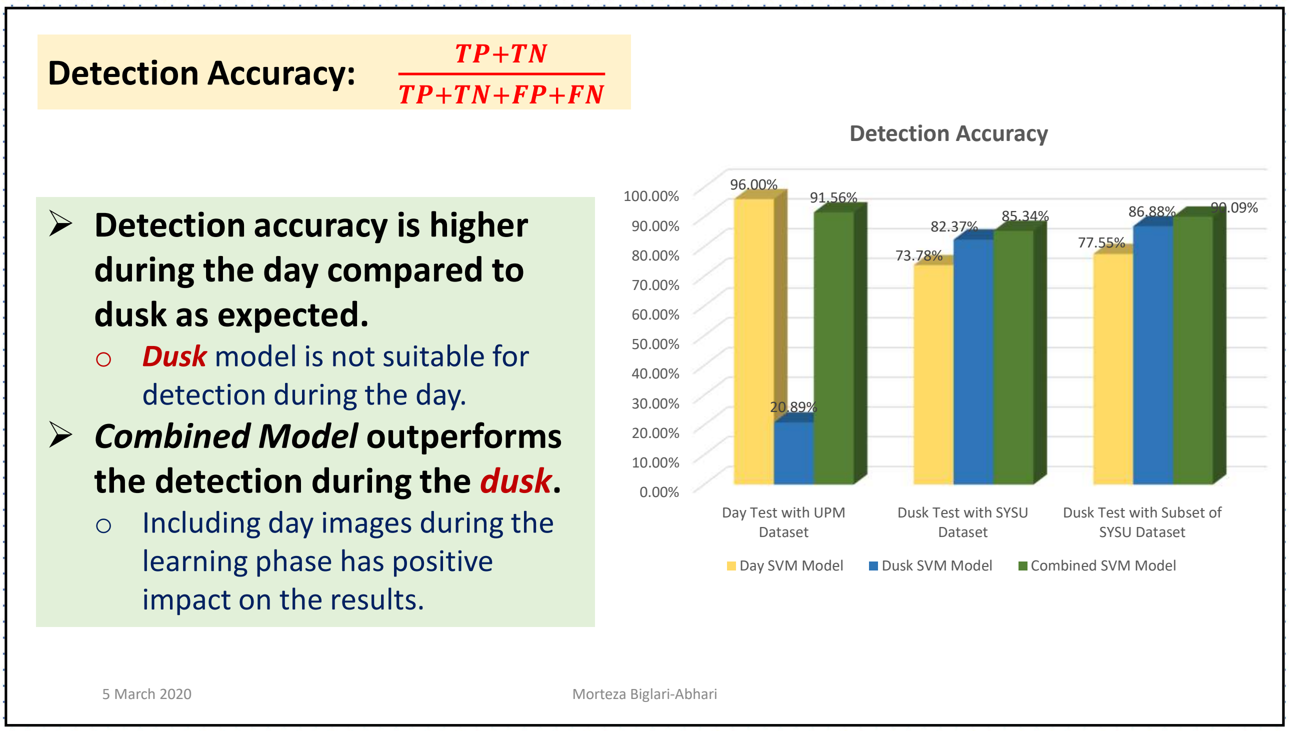
We developed an adaptive vehicle detection approach for day, dusk and dark.

Vehicle Detection- Day and Dusk:

Different training datasets are used for *Day* and *Dusk*:

- Two different models (separate datasets) for SVM classification
- **Combined Model:** Third model generated by training the classifier with both of the ***Day*** and ***Dusk*** datasets together.

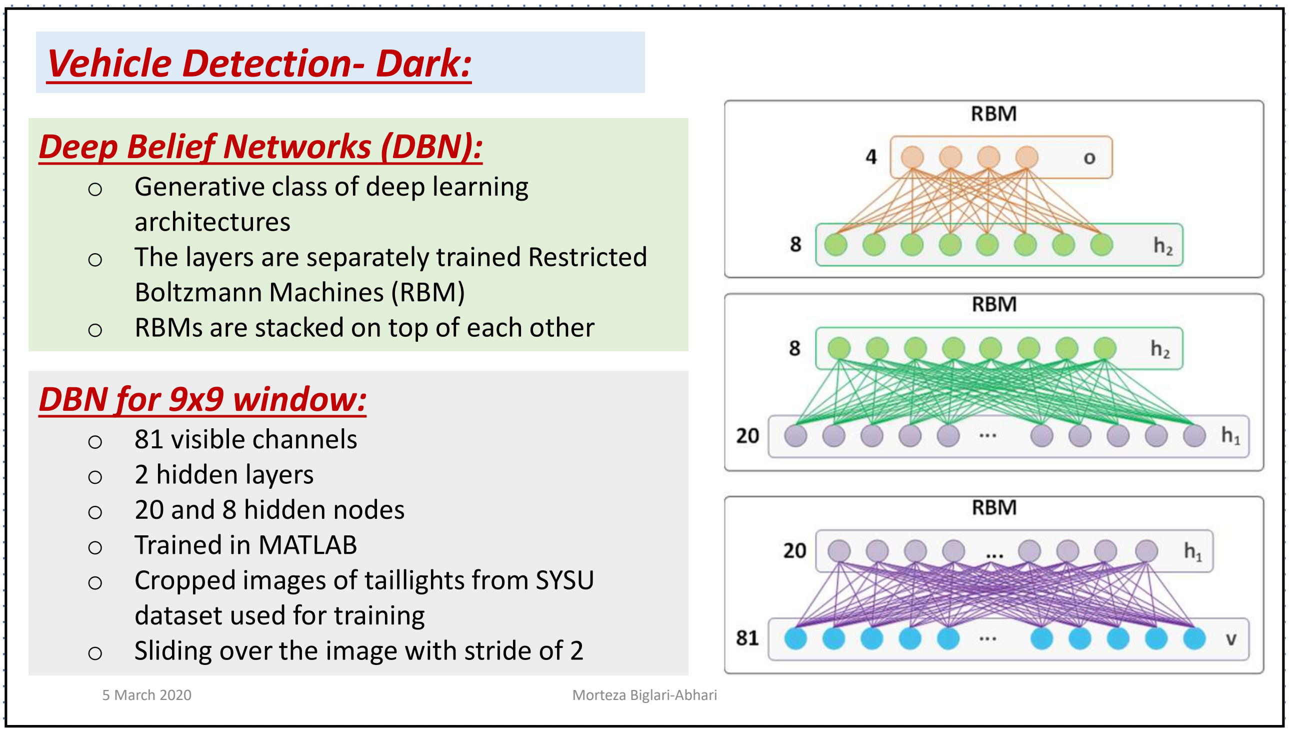




Detection Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$

- **Detection accuracy is higher during the day compared to dusk as expected.**
  - ○ ***Dusk*** model is not suitable for detection during the day.
- ***Combined Model*** **outperforms the detection during the** ***dusk*****.**
  - ○ Including day images during the learning phase has positive impact on the results.

5 March 2020 Morteza Biglari-Abhari



Vehicle Detection- Dark:

Deep Belief Networks (DBN):

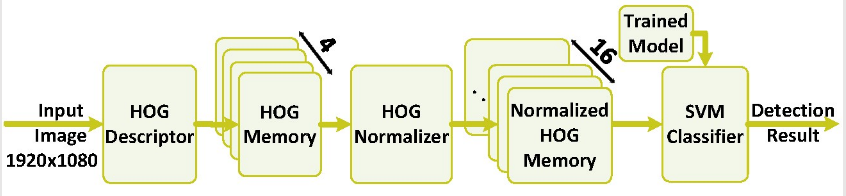
- ○ Generative class of deep learning architectures
- ○ The layers are separately trained Restricted Boltzmann Machines (RBM)
- ○ RBMs are stacked on top of each other

DBN for 9x9 window:

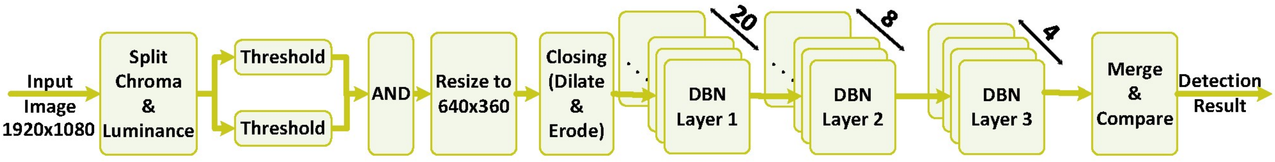
- ○ 81 visible channels
- ○ 2 hidden layers
- ○ 20 and 8 hidden nodes
- ○ Trained in MATLAB
- ○ Cropped images of taillights from SYSU dataset used for training
- ○ Sliding over the image with stride of 2

5 March 2020 Morteza Biglari-Abhari

Summary of our Vehicle Detection approach:



Detection rate of 60 fps for an HDTV frame in Day, and Dusk



Detection rate of 50 fps for an HDTV frame in Dark

Dynamic Partial Reconfiguration:

Partial reconfiguration **(PR) is an advanced feature of FPGAs for *run-time resource management*:**

- **Time-multiplexing hardware resources**
- **Flexibility of SW with performance of HW**
- **Reconfiguration time and overhead is the concern**

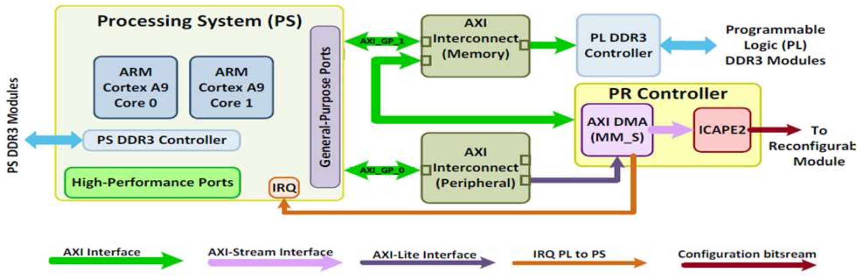
Partial reconfiguration throughput in ZYNQ SoC:

- **Theoretical value of *400MB/sec* at working frequency of *100MHz***
- **Limited to only *19MB/sec* for ICAP**
  - **Bitstreams transfer through general purpose ports of PS to AXI_HWICAP**
- **Limited to *145MB/sec* for *PCAP***
  - **Affected by Zynq central interconnect delays**

Delays in Partial Reconfiguration are related to the connection ports (PS through GP ports, DMA core to PL DDR, availability of AXI HP ports)

We improved the reconfiguration throughput to *390MB/sec*:

- ➢ **Measured in PS by ARM performance event counters**
- ➢ **Measured in PL by Vivado integrated logic analyzer (ILA)**

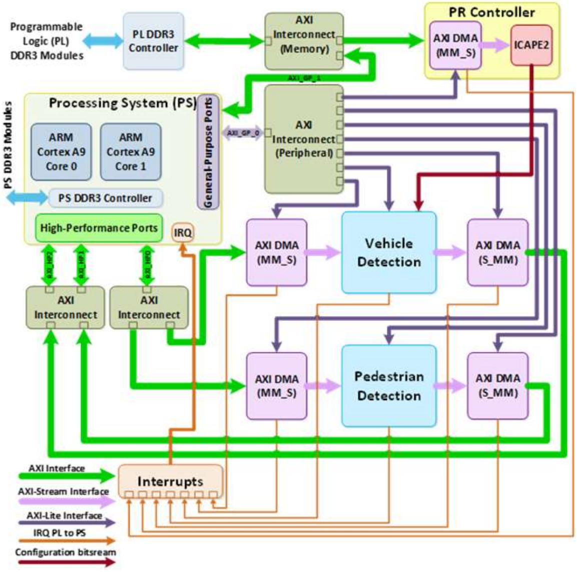


Block diagram of the connection between PL DDR3, PR controller, and Zynq PS

5 March 2020 Morteza Biglari-Abhari

Summary: Adaptive Vehicle/Pedestrian Detection for ADS

- ➢ **Static Region**
  - ○ **Pedestrian Detection**
  - ○ **PL to PS Connections**
  - ○ **PR Controller**
- ➢ **Reconfigurable Region**
  - ○ **Vehicle Detection**
  - ○ **Two different configurations**
- ➢ **Transition between dusk and dark requires the reconfiguration**
  - ○ **Does not happen frequently**
  - ○ **Tunnel environment is well lighted and is categorized as dusk.**
  - ○ **Scenarios such as entering the tunnel is handled by the switching between day and dusk**



5 March 2020 Morteza Biglari-Abhari

Security Vulnerabilities

Security issues are related to:

Confidentiality: **to prevent information access or leakage to unauthorized parties**
(e.g. using cryptographic algorithms such as AES)

Integrity: **to ensure that the information has not been tampered with**
(e.g. using cryptographic hashing such as SHA-2)

Authentication: **to ensure that the information can be available only to identified parties and whenever they need**
(e.g using a combination of encryption and hashing (based on public keys or shared keys)

○ **We are concerned with the security vulnerabilities of MPSoC.**

Examples of Security Threats

Physical Hardware attacks:

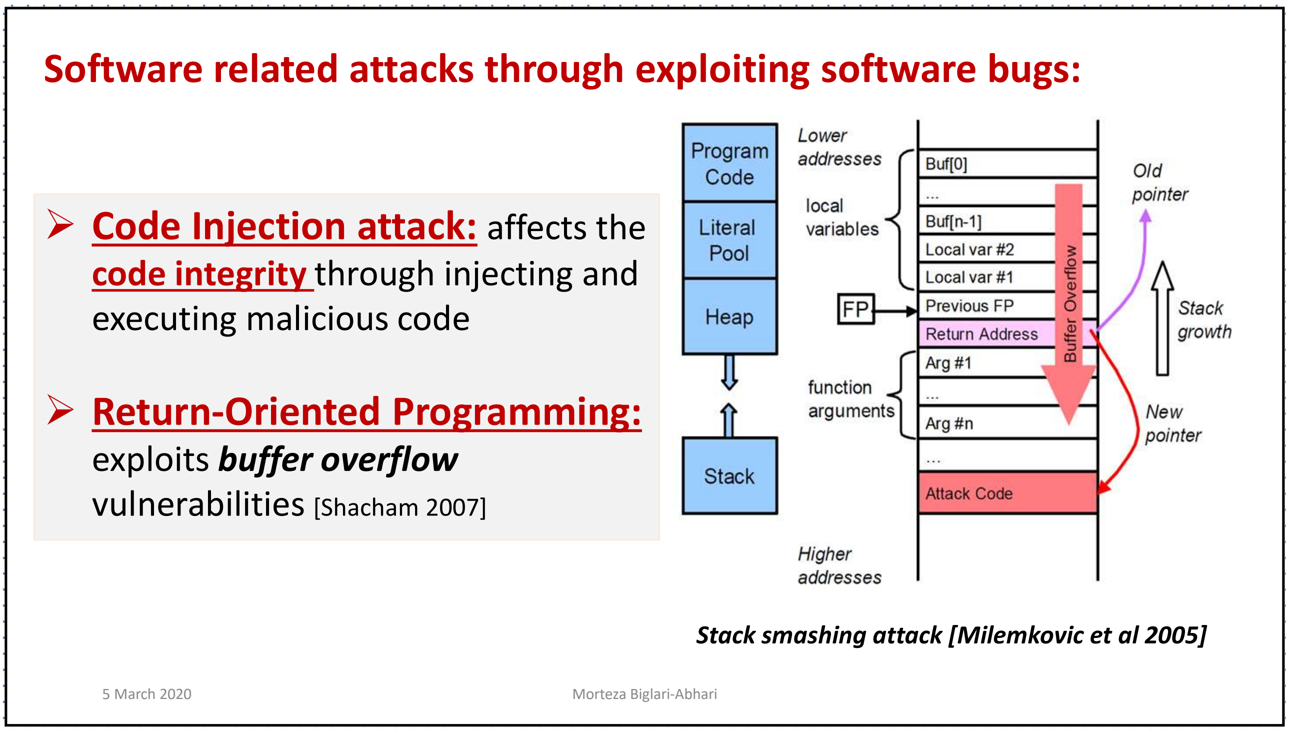
➢ **Invasive:** Physical manipulation of hardware devices (the so-called "shack attacks")

➢ **Non-Invasive:**

- ○ Tampering with the device functionality (through debug interfaces e. g. JTAG or USB ports)
- ○ Side-channel attacks to extract secret information (usually cryptographic keys, other private or valuable information) through a covert side-channel

Hardware Trojan: a malicious hardware component or IP embedded in the system to expose secret information.

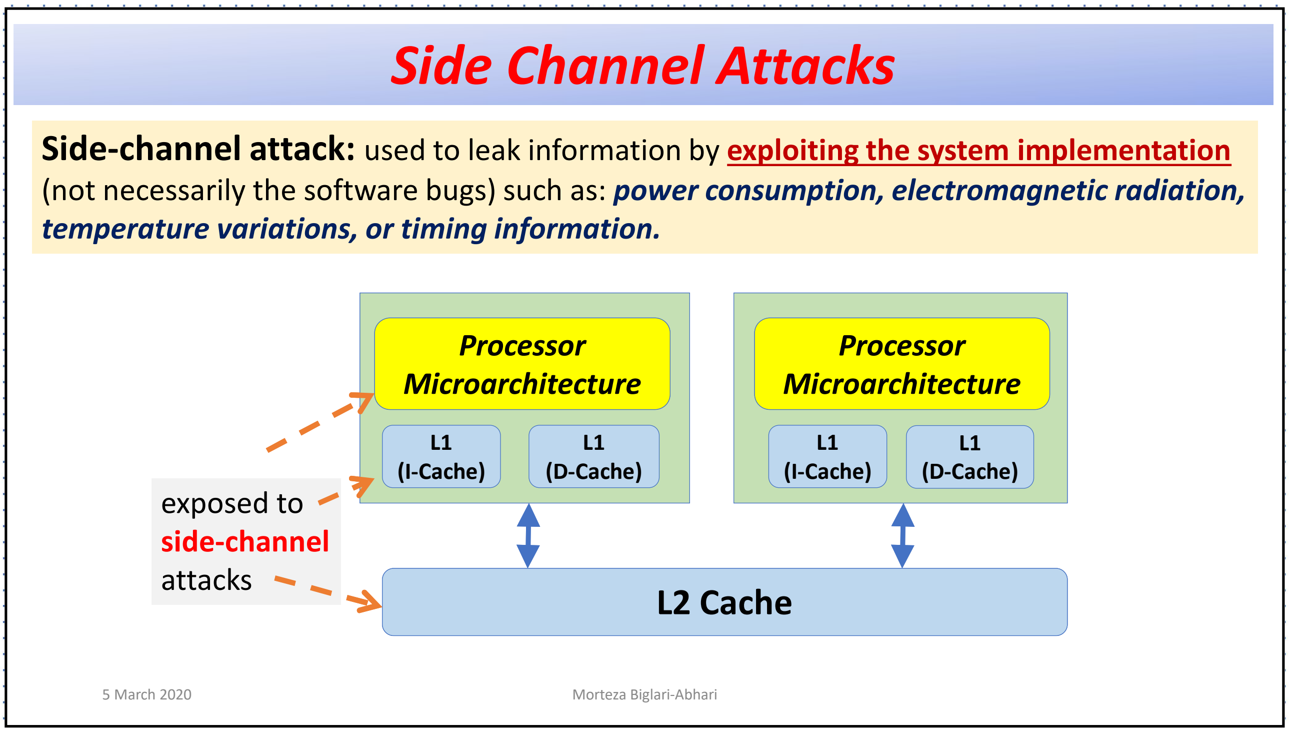
IP Stealing: mitigated by using Physically Unclonable Function (PUF) technology



Software related attacks through exploiting software bugs:

- **Code Injection attack:** affects the **code integrity** through injecting and executing malicious code
- **Return-Oriented Programming:** exploits ***buffer overflow*** vulnerabilities [Shacham 2007]

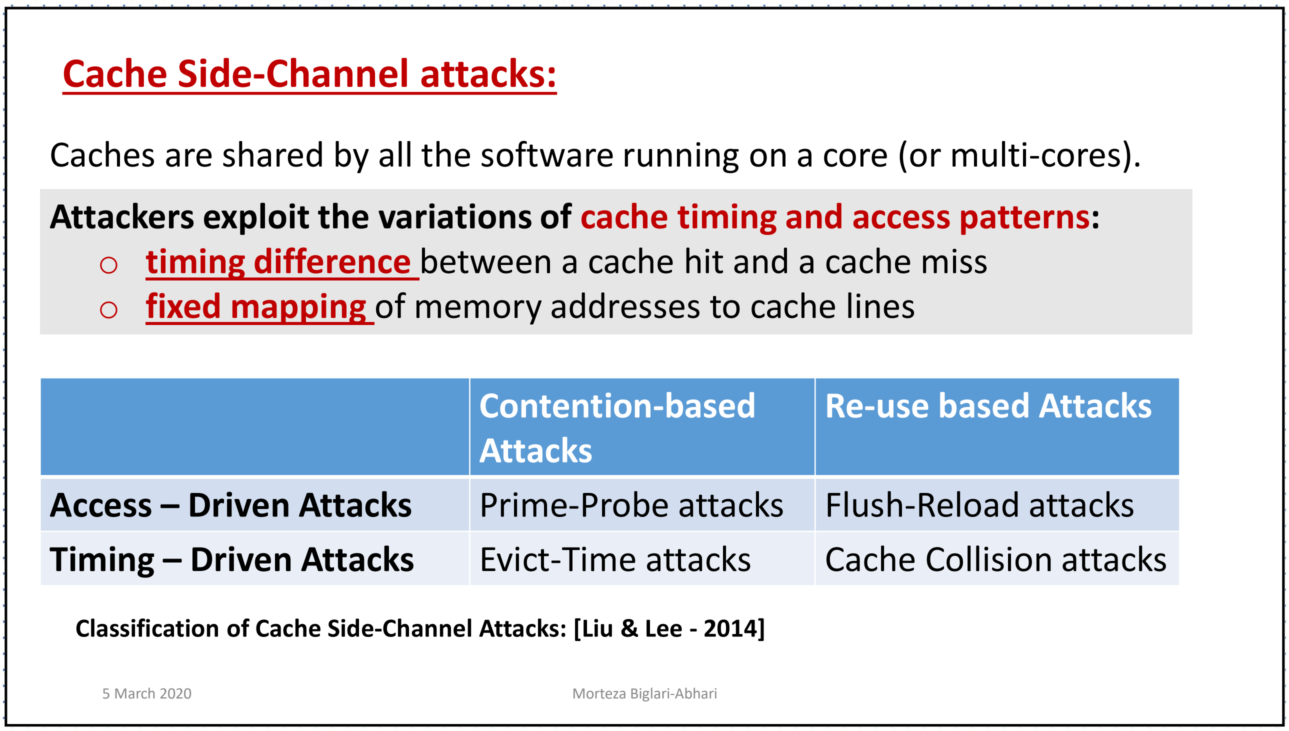
Stack smashing attack [Milemkovic et al 2005]



Side Channel Attacks

Side-channel attack: used to leak information by exploiting the system implementation (not necessarily the software bugs) such as: ***power consumption, electromagnetic radiation, temperature variations, or timing information.***

Processor Microarchitecture
L1 (I-Cache)
L1 (D-Cache)
Processor Microarchitecture
L1 (I-Cache)
L1 (D-Cache)
L2 Cache
exposed to side-channel attacks



Cache Side-Channel attacks:

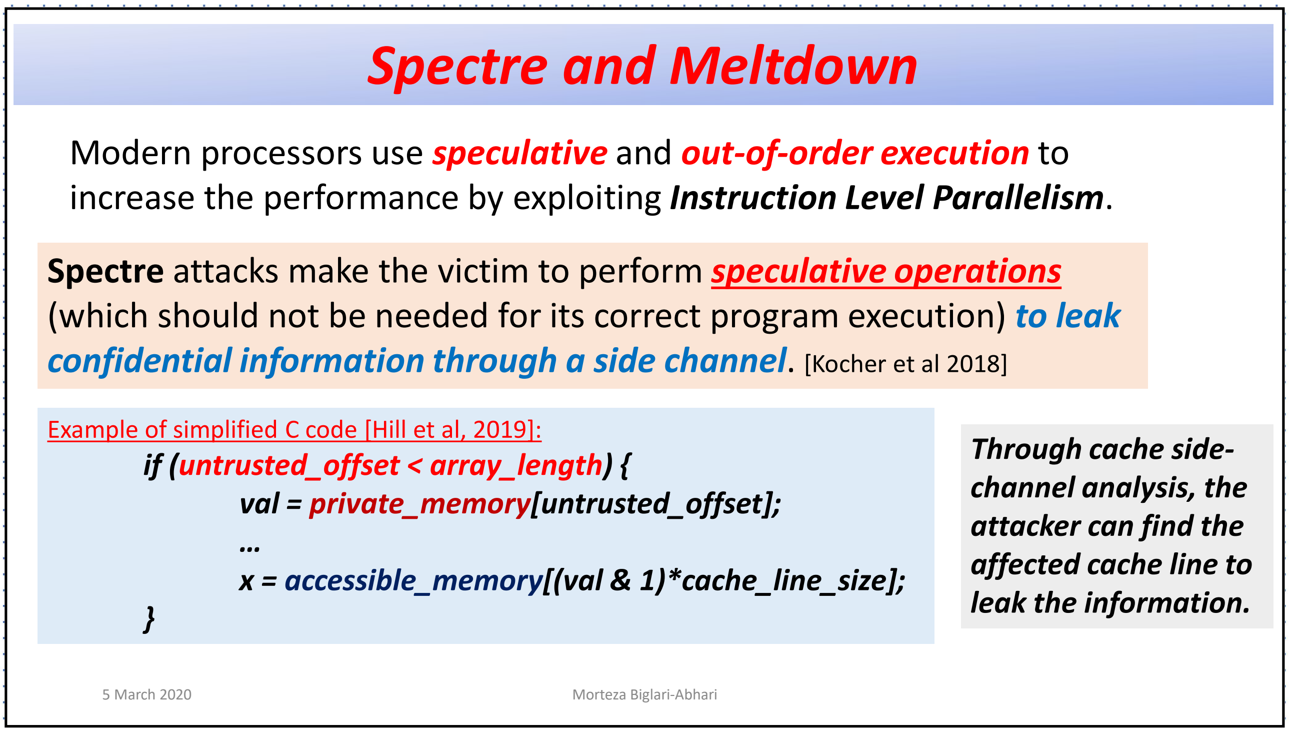
Caches are shared by all the software running on a core (or multi-cores).

Attackers exploit the variations of cache timing and access patterns:

- timing difference between a cache hit and a cache miss
- fixed mapping of memory addresses to cache lines

	Contention-based Attacks	Re-use based Attacks
Access – Driven Attacks	Prime-Probe attacks	Flush-Reload attacks
Timing – Driven Attacks	Evict-Time attacks	Cache Collision attacks

Classification of Cache Side-Channel Attacks: [Liu & Lee - 2014]



Spectre and Meltdown

Modern processors use ***speculative*** and ***out-of-order execution*** to increase the performance by exploiting ***Instruction Level Parallelism***.

Spectre attacks make the victim to perform ***speculative operations*** (which should not be needed for its correct program execution) ***to leak confidential information through a side channel***. [Kocher et al 2018]

Example of simplified C code [Hill et al, 2019]:

```
        if (untrusted_offset < array_length) {
                val = private_memory[untrusted_offset];
                …
                x = accessible_memory[(val & 1)*cache_line_size];
        }
```

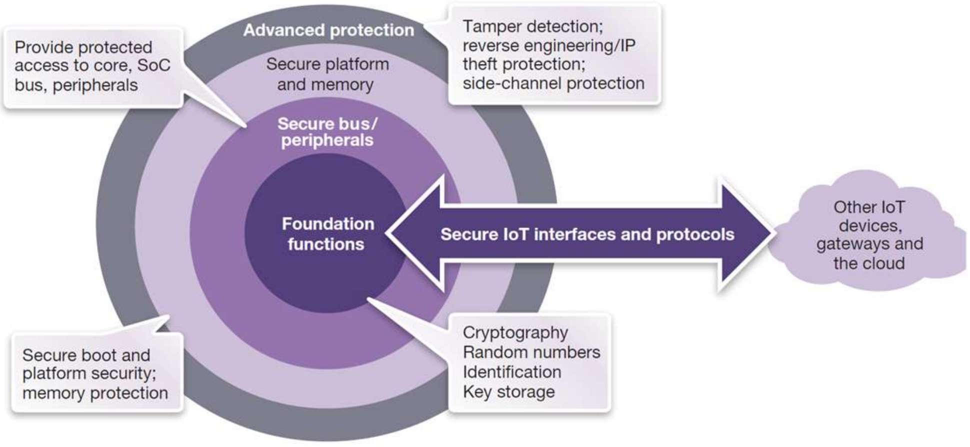
Through cache side-channel analysis, the attacker can find the affected cache line to leak the information.

- **Meltdown** attack exploits ***out-of-order execution*** to leak the contents of part of the physical memory.
- **Meltdown** exploits a ***privilege escalation vulnerability*** which is specific to Intel processors (so memory protection can be bypassed by speculatively executed instructions) [Lipp et al - 2018].

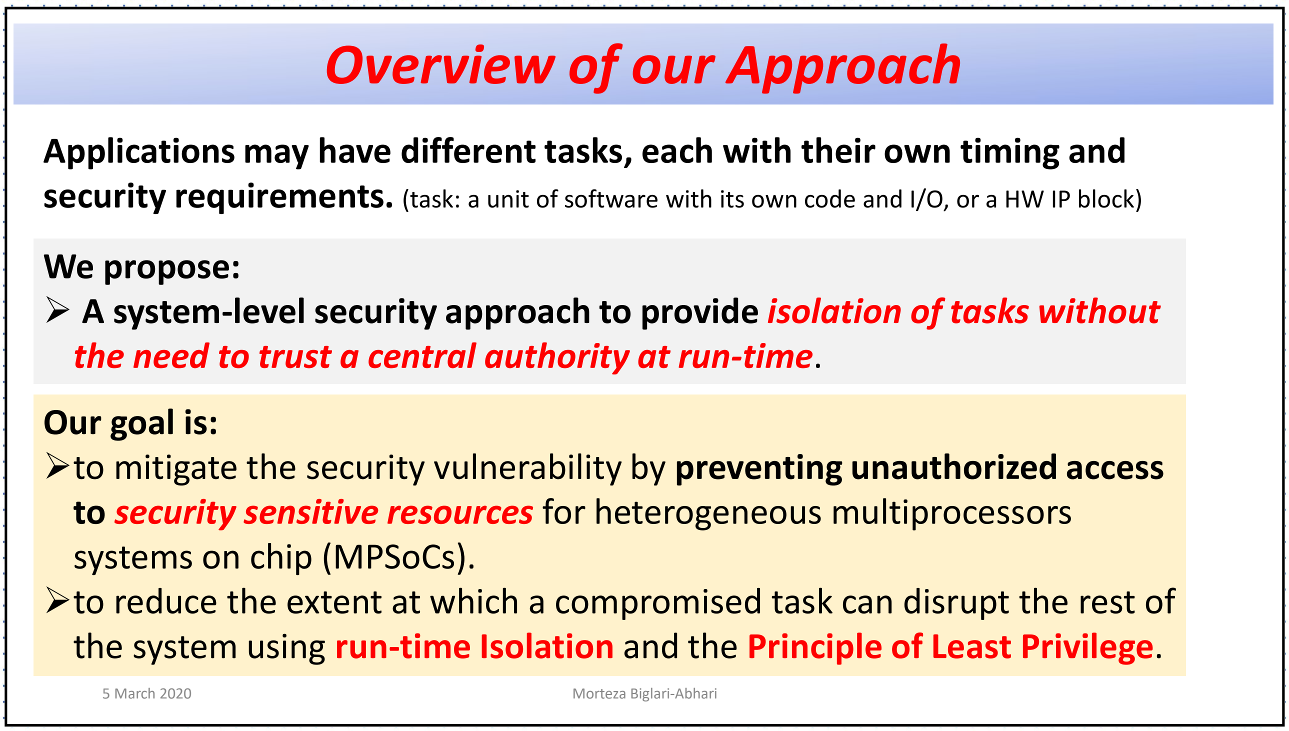
- ○ **Forshadow** attacks (reported a few months after **Spectre** and **Meltdown**) are specificly target Intel Processors.
- ○ **Initial versions targeted SGX enclave data. Newer versions target kernel memory (in OS, Virtual Machines and Hypervisors).** [Kan 2018]

Addressing Security Vulnerabilities

Layered Security Support


Advanced protection
Secure platform and memory
Secure bus/ peripherals
Foundation functions
Secure IoT interfaces and protocols
Other IoT devices, gateways and the cloud
Provide protected access to core, SoC bus, peripherals
Tamper detection; reverse engineering/IP theft protection; side-channel protection
Secure boot and platform security; memory protection
Cryptography
Random numbers
Identification
Key storage

Overview of IoT Security Solutions (source: white paper (March 2016) - synopsys.com)



Overview of our Approach

Applications may have different tasks, each with their own timing and security requirements. (task: a unit of software with its own code and I/O, or a HW IP block)

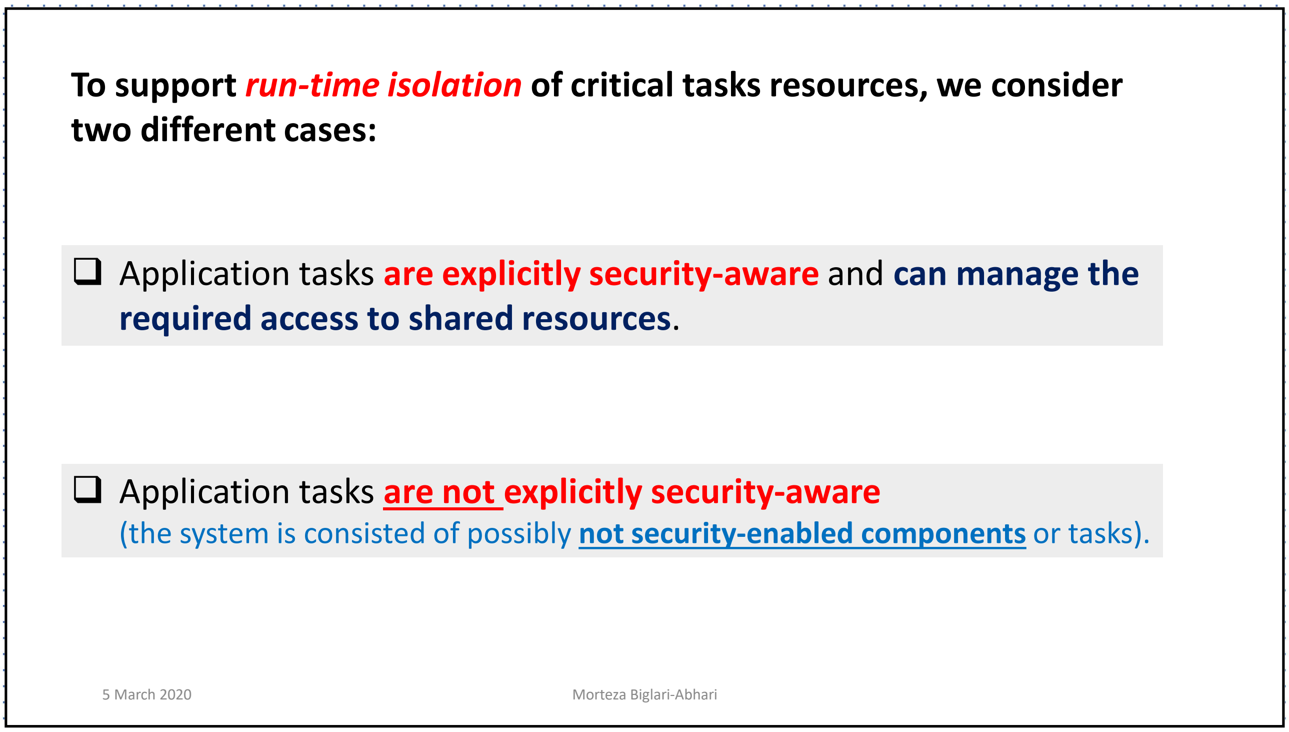
We propose:

➢ **A system-level security approach to provide *isolation of tasks without the need to trust a central authority at run-time*.**

Our goal is:

➢ to mitigate the security vulnerability by **preventing unauthorized access to *security sensitive resources*** for heterogeneous multiprocessors systems on chip (MPSoCs).

➢ to reduce the extent at which a compromised task can disrupt the rest of the system using **run-time Isolation** and the **Principle of Least Privilege**.

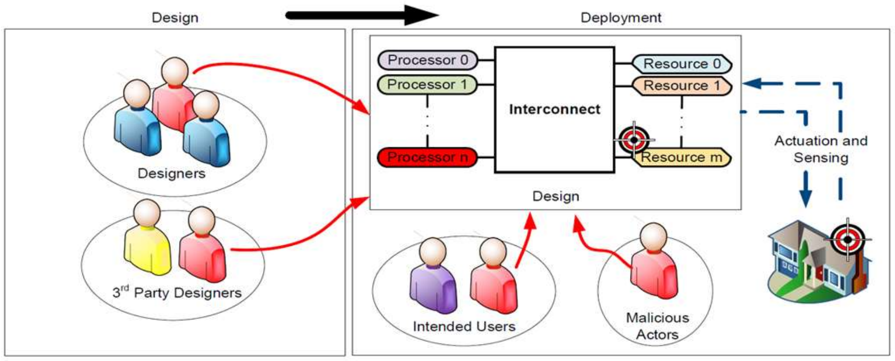


To support *run-time isolation* of critical tasks resources, we consider two different cases:

❑ Application tasks **are explicitly security-aware** and **can manage the required access to shared resources**.

❑ Application tasks **are not explicitly security-aware**
(the system is consisted of possibly **not security-enabled components** or tasks).

Application Example: (Integrated Home Automation Hub)



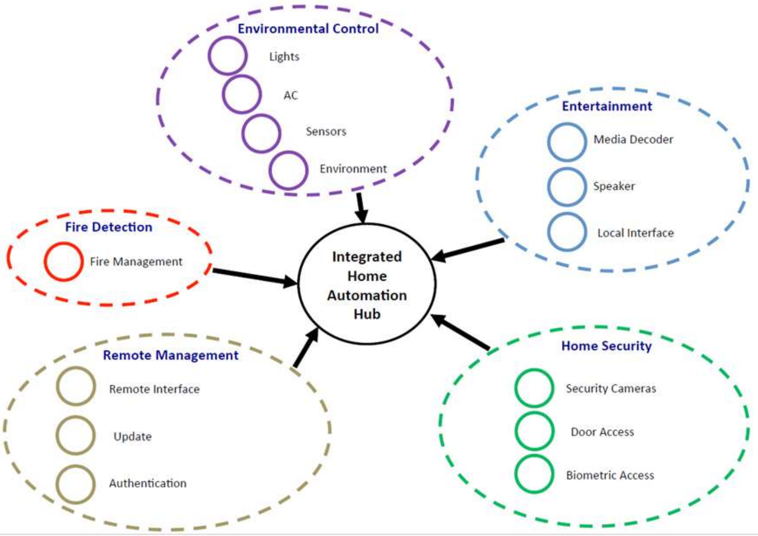
Various **Sensors** and **Actuators** *(different levels of criticality)*

Security threats may affect correct *functionality* and *information flow* in the system.

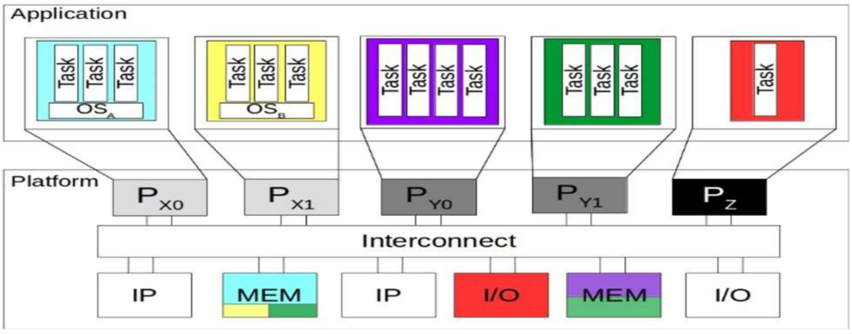
Collection of *tasks* in an Integrated Home Automation Hub:

Tasks:

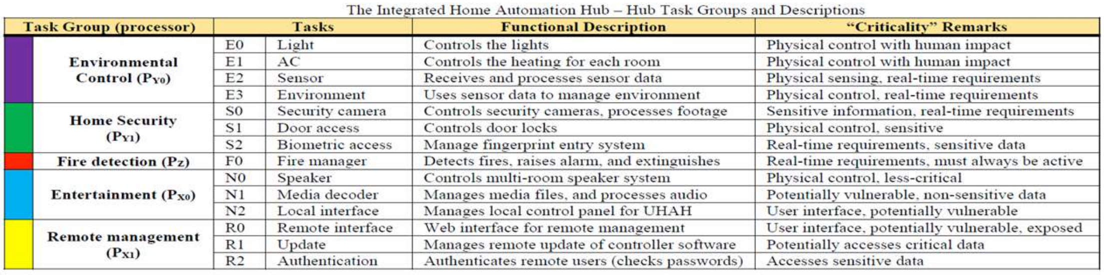
- run concurrently
- are grouped based on their functionality
- may interact with other tasks in the same group or tasks in other groups



Tasks allocation:

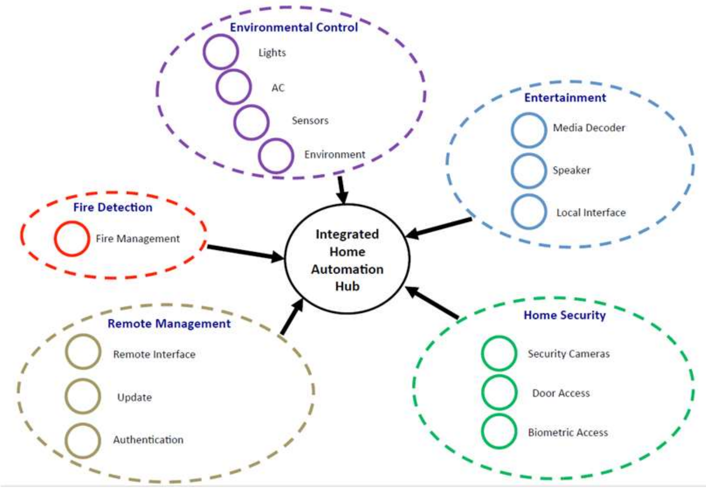


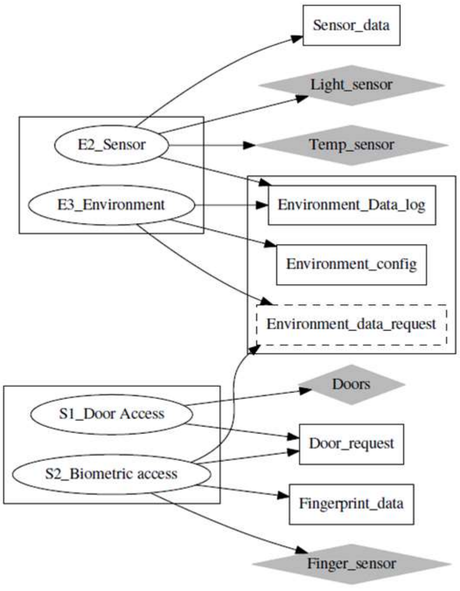
The Integrated Home Automation Hub – Hub Task Groups and Descriptions



Task Group (processor)		Tasks		Functional Description	"Criticality" Remarks
🟪	Environmental Control (P_{Y0})	E0	Light	Controls the lights	Physical control with human impact
		E1	AC	Controls the heating for each room	Physical control with human impact
		E2	Sensor	Receives and processes sensor data	Physical sensing, real-time requirements
		E3	Environment	Uses sensor data to manage environment	Physical control, real-time requirements
🟩	Home Security (P_{Y1})	S0	Security camera	Controls security cameras, processes footage	Sensitive information, real-time requirements
		S1	Door access	Controls door locks	Physical control, sensitive
		S2	Biometric access	Manage fingerprint entry system	Real-time requirements, sensitive data
🟥	Fire detection (P_Z)	F0	Fire manager	Detects fires, raises alarm, and extinguishes	Real-time requirements, must always be active
🟦	Entertainment (P_{X0})	N0	Speaker	Controls multi-room speaker system	Physical control, less-critical
		N1	Media decoder	Manages media files, and processes audio	Potentially vulnerable, non-sensitive data
		N2	Local interface	Manages local control panel for UHAH	User interface, potentially vulnerable
🟨	Remote management (P_{X1})	R0	Remote interface	Web interface for remote management	User interface, potentially vulnerable, exposed
		R1	Update	Manages remote update of controller software	Potentially accesses critical data
		R2	Authentication	Authenticates remote users (checks passwords)	Accesses sensitive data

Example (Interaction among task groups):





How to reduce the attack surface?

Simple T/R Graph: *Diamonds: physical resources*
Rectangles: memory

Threat Model

MPSoCs typically have two main risks:

- Shared resources, accessible between different task groups
- Shared interconnect, which typically offer all-to-all access

We assume that an attacker can compromise a task in its entirety so a compromised task:

- → can attempt to generate a memory access to any part of the platform
- → can configure a DMA-capable IP to access a part of memory on its behalf
- → that runs over an OS, which can escalate its privileges, and disable any existing **Memory Protection Unit**.

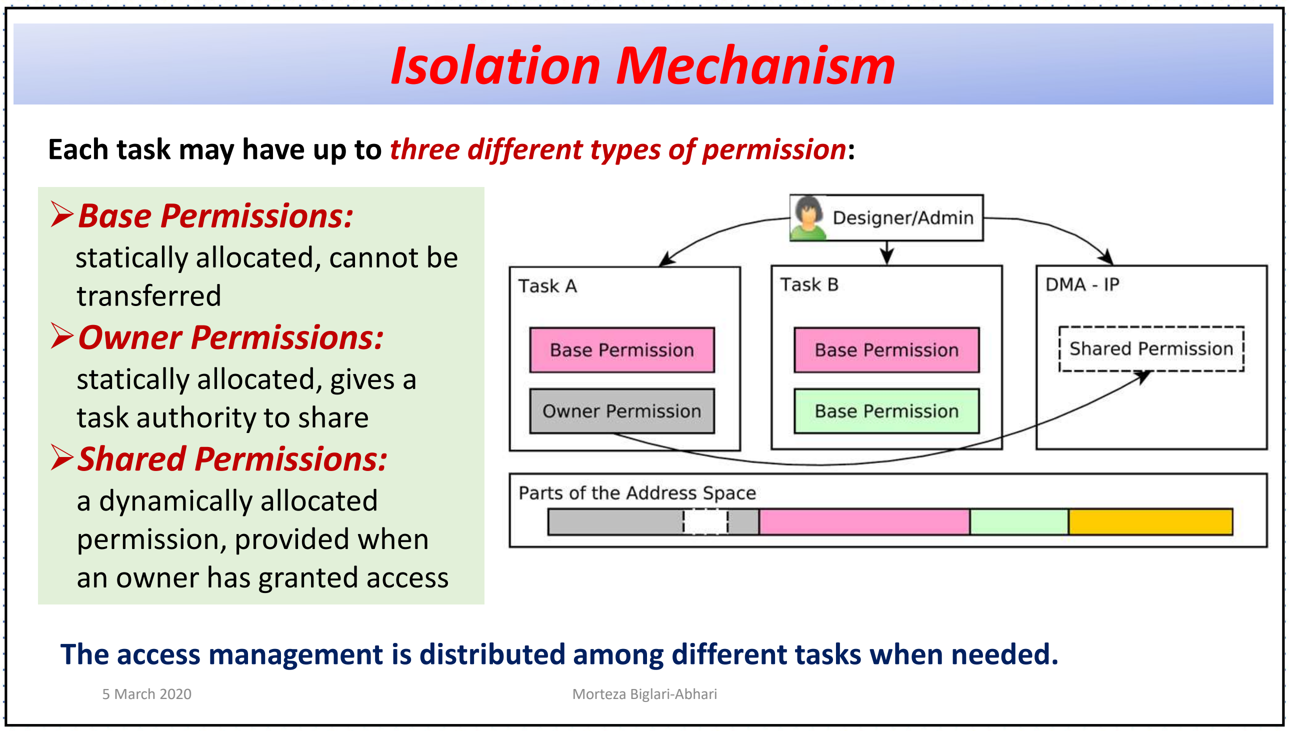
(Or potentially, reconfigure the MPU to disrupt other tasks running on that processor)

Security rules should specify how *shared data*, *shared code* and *IP blocks* should be managed.

- ◊ Tasks ***manage their own accessible resources***.
- ◊ To reduce the impact of a compromised task, ***memory accesses are regulated through dynamic access permission setup***.

How can we implement access control in our system?

- → **Mandatory Access Control (MAC):** a centralized and privileged administrator manages all access permissions
- → **Discretionary Access Control (DAC):** entities grant and revoke access to objects they own to each other
- → **Role-based Access Control (RAC):** access permissions are attached not to tasks, but to "jobs" or "functions" that a task may be performing

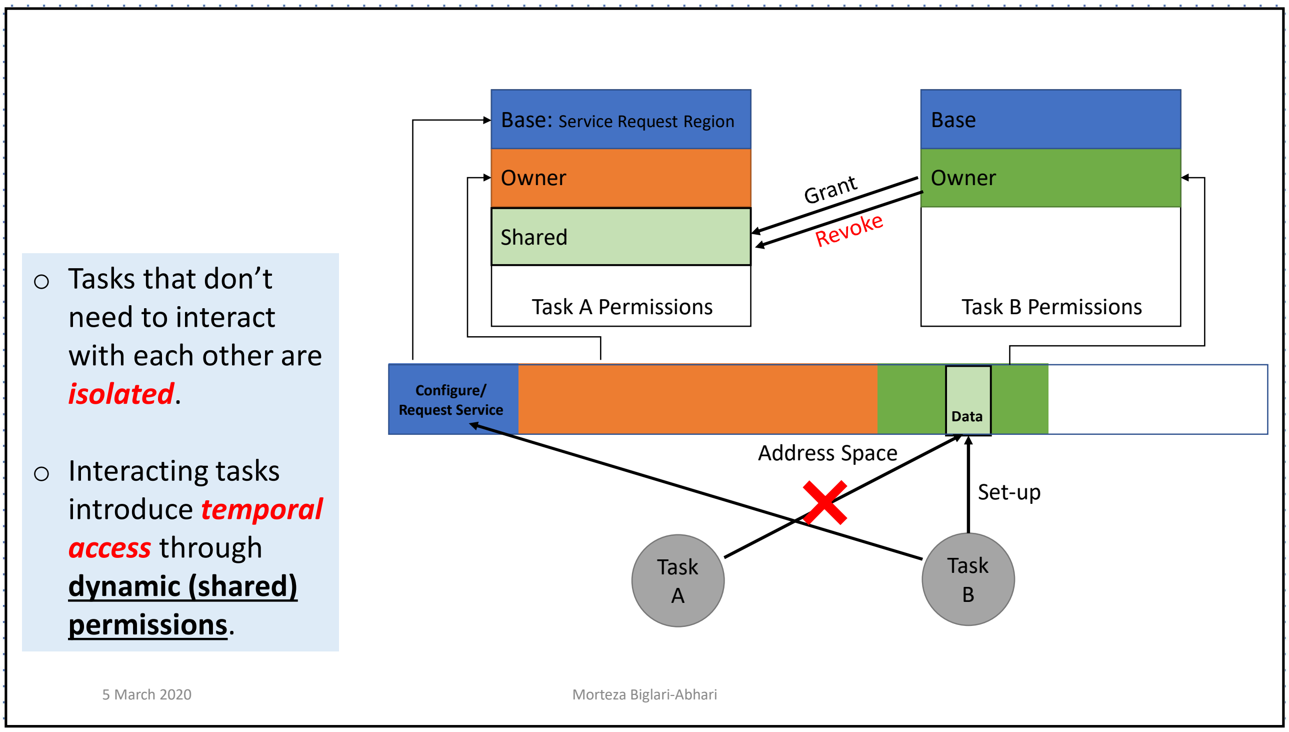


Isolation Mechanism

Each task may have up to *three different types of permission*:

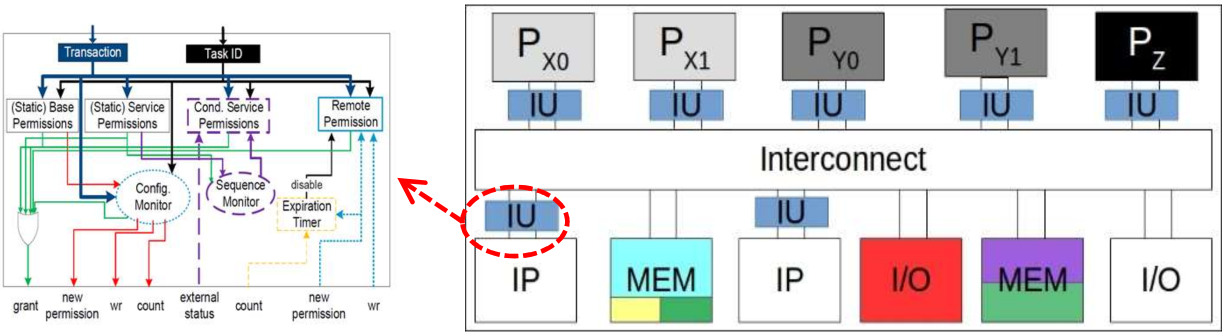
- ***Base Permissions:*** statically allocated, cannot be transferred
- ***Owner Permissions:*** statically allocated, gives a task authority to share
- ***Shared Permissions:*** a dynamically allocated permission, provided when an owner has granted access

The access management is distributed among different tasks when needed.



- ○ Tasks that don't need to interact with each other are ***isolated***.
- ○ Interacting tasks introduce ***temporal access*** through **dynamic (shared) permissions**.

Isolation Unit (IU):

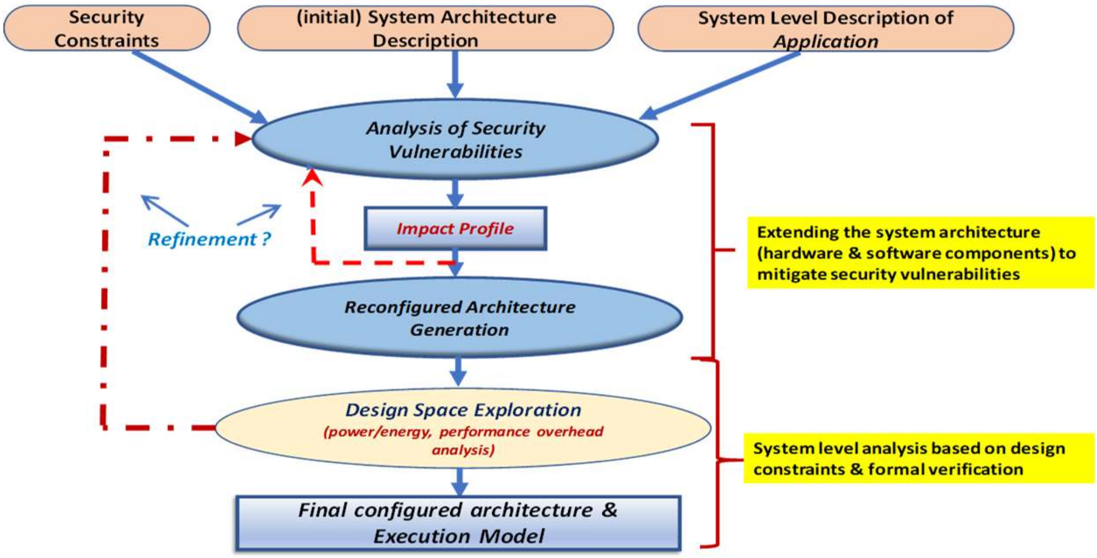


- ➢ **IU**s check ***memory transactions*** at each processor/IP block
- ➢ Each **IU** is ***memory-mapped*** into the address space of the local processor
- ➢ **IU**s can be interconnected using point-to-point connections, or a separate dedicated interconnect structure.

However, the *tasks may not be security-aware* as IP may be provided from different sources (e.g. shared libraries, third party IPs etc). So, the previous approach extended to:

- ➢ **Providing a mechanism for designer to specify the *required security rules***
- ➢ **Providing architecture level support *to reduce the impact* of the compromised component(s) on other parts of the system**
  - ○ **Automatically isolate the critical parts of the systems**
  - ○ **Use a proper interfacing between the critical and non-critical parts**
  - ○ **Generate a new (security optimized) system architecture**
  - ○ **Analyze the overhead (performance, power/energy) of the security-enhanced architecture**

Security-aware System Design Flow



Security Constraints

(initial) System Architecture Description

System Level Description of *Application*

Analysis of Security Vulnerabilities

Impact Profile

Refinement ?

Reconfigured Architecture Generation

Design Space Exploration

(power/energy, performance overhead analysis)

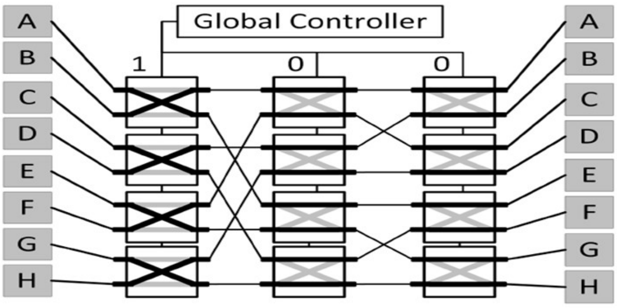
Final configured architecture & Execution Model

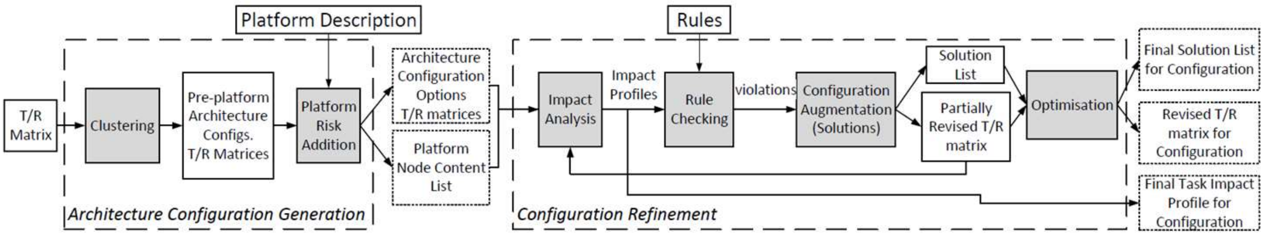
Extending the system architecture (hardware & software components) to mitigate security vulnerabilities

System level analysis based on design constraints & formal verification

Case Study (Integrated Home Automation Hub):

- based on TDMA-MIN NoC platform with 8 nodes
- TDMA scheme establishes temporal isolation





Platform Description

Rules

T/R Matrix

Clustering

Pre-platform Architecture Configs. T/R Matrices

Platform Risk Addition

Architecture Configuration Options T/R matrices

Platform Node Content List

Architecture Configuration Generation

Impact Analysis

Impact Profiles

Rule Checking

violations

Configuration Augmentation (Solutions)

Solution List

Partially Revised T/R matrix

Optimisation

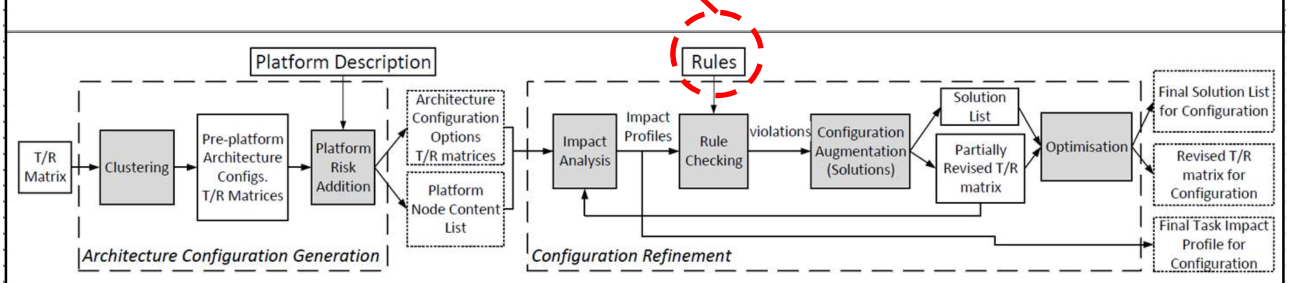
Configuration Refinement

Final Solution List for Configuration

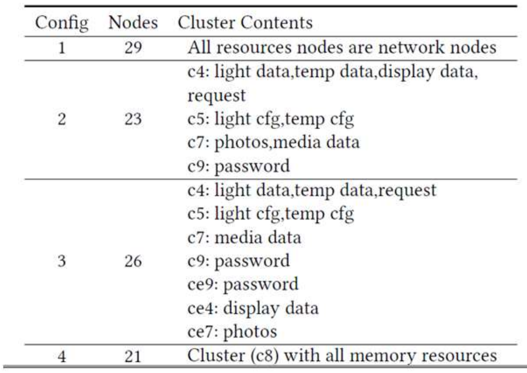
Revised T/R matrix for Configuration

Final Task Impact Profile for Configuration

- **Type 1 (Impact restriction):** to specify that a task asset is not “impacted” by another task
- **Type 2 (Resource class access restriction):** to specify when a task should not have access to resource assets with specific attributes
- **Type 3 (Specific resource restriction):** same as Type 2, except for a specific resource asset
- **Type 4 (Untrusted task):** to specify that a specific task may not be trustworthy
- **Type 5 (Resource exclusivity):** complement of Type 3, to ensure that only one task can write to a resource

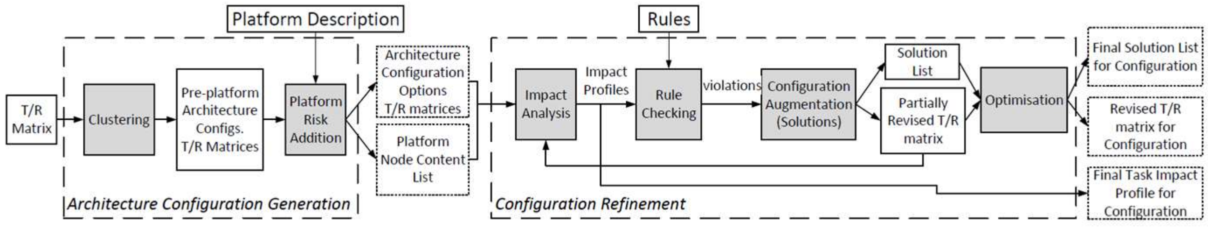

Platform Description
Rules
T/R Matrix
Clustering
Pre-platform Architecture Configs. T/R Matrices
Platform Risk Addition
Architecture Configuration Options T/R matrices
Platform Node Content List
Architecture Configuration Generation
Impact Analysis
Impact Profiles
Rule Checking
violations
Configuration Augmentation (Solutions)
Solution List
Partially Revised T/R matrix
Optimisation
Configuration Refinement
Final Solution List for Configuration
Revised T/R matrix for Configuration
Final Task Impact Profile for Configuration

Configuration Results:



Config	Nodes	Cluster Contents
1	29	All resources nodes are network nodes
2	23	c4: light data,temp data,display data, request c5: light cfg,temp cfg c7: photos,media data c9: password
3	26	c4: light data,temp data,request c5: light cfg,temp cfg c7: media data c9: password ce9: password ce4: display data ce7: photos
4	21	Cluster (c8) with all memory resources

Config.	Total port checks	Total MPU permissions	No. IUs	Resource overhead
1	19	4	4	+10.7%
2	14	5	2	+7.3%
3	15	4	2	+6.5%
4	9	18	1	+12.2%


Platform Description
Rules
T/R Matrix
Clustering
Pre-platform Architecture Configs. T/R Matrices
Platform Risk Addition
Architecture Configuration Options T/R matrices
Platform Node Content List
Architecture Configuration Generation
Impact Analysis
Impact Profiles
Rule Checking
violations
Configuration Augmentation (Solutions)
Solution List
Partially Revised T/R matrix
Optimisation
Configuration Refinement
Final Solution List for Configuration
Revised T/R matrix for Configuration
Final Task Impact Profile for Configuration

Some Related Works

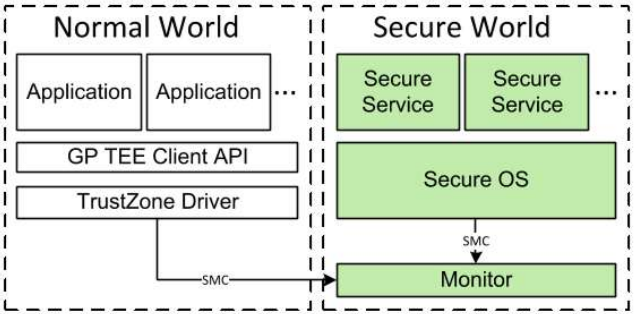
- Attack detection using hardware monitors [Patel et al - 2011]
- Creating on-chip sandboxes [Bathen & Dutt - 2010]
- Security-aware on-chip interconnection network:
  - NoC firewalls [Fiorin et al - 2008, LeMay & Gunter - 2014, Grammatikakis et al - 2015]

  Typically rely on OS or a central authority for managing rules at run-time. Also rules are not changed dynamically.
- TrustLite: Extension of MPU as Execution-Aware Memory Protection Unit (EA-MPU), where tasks memory access permissions are stored in Centralized Tables [Koeberl et al. - 2014]
- TyTAN: An extension of TrustLite for tiny embedded devices by run-time task loading using a secure RTOS [Brasser et al. - 2015]

ARM TrustZone [2004]:

- The processor core is divided into two parts **secure world** and **normal world**

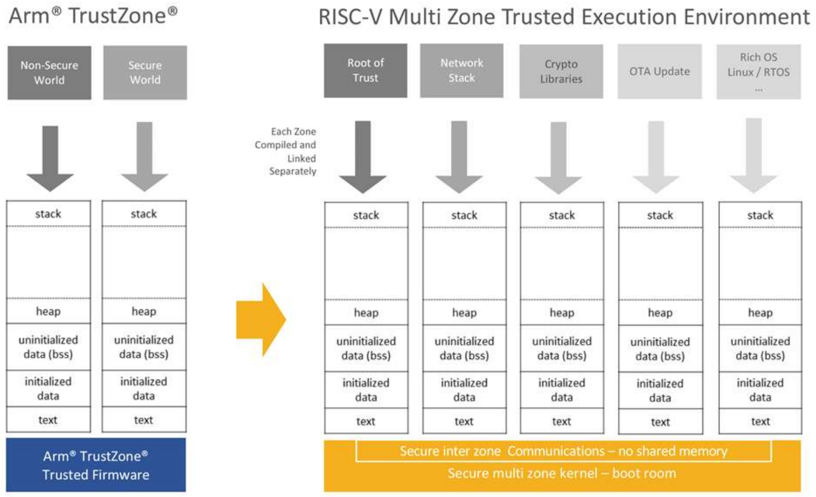
- Since it only offers two "worlds", cannot easily be used in multi-core systems
- Interaction between the two worlds are managed differently in ARMv7 and ARMv8 (for Cortex-M processors).



- **ARM TrustZone** has been extended for Cortex-M processors in ARMv8-M.
- Unlike Cortex-A processors, the division of Secure and Normal worlds ***is memory map based*** (transitions takes place automatically in exception handling code, and multiple secure entry points are supported.

RISC-V MultiZone [Si-Five: 2019]:

- **Software Defined MultiZone**
- Different number of zones can be allocated ***statically*** based on the application needs
- Architectural support is through **Physical Memory Protection (PMP)**



Source: Cesar Garlati: (Hex5 Security), 2019

Conclusions

- We developed hardware accelerator for an adaptive vehicle and pedestrian detection using ***dynamic partial reconfiguration*** on FPGAs where, different detection algorithms may be used for different environment conditions.

- We proposed a **System-Level design flow for security enhanced** MPSoC for Cyber-Physical systems:
  - To specify the required security rules
  - Automatically isolate the critical parts of the systems
  - Use a proper interfacing between the critical and non-critical parts
  - Generate a new (security optimized) system architecture

Our future works are related to: Analyzing the overhead (performance, power/energy, time predictability) of the security-enhanced architecture (with design space exploration), RISC-V microarchitecture.

Our related papers:

- Hemmati, M., Biglari-Abhari, M., & Niar, S. (**DATE - 2019**) *Adaptive Vehicle Detection for Real-time Autonomous Driving System*, in Proceedings of the 2019 IEEE Conference on Design, Automation & Test in Europe (DATE), Florence, Italy, 25-28 March 2019, pp. 1034-1039
- Hemmati, M., Biglari-Abhari, M., Niar, S., & Berber, S. (**DAC - 2017**) *Real-Time Multi-Scale Pedestrian Detection for Driver Assistance Systems*. In Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference 2017, pp. 1-6, Austin, TX, US
- Tan, B., Biglari-Abhari, M., Salcic, Z., (**ACM-TECS - 2017**) *An Automated Security-Aware Approach for Design of Embedded Systems on MPSoC*, ACM Transactions on Embedded Computing Systems (TECS), October 2017
- Tan, B., Biglari-Abhari, M., Salcic, Z., (**JSA - 2017**) *Towards Decentralized System-Level Security for MPSoC-based Embedded Applications*, Journal of Systems Architecture, Volume 80, Oct. 2017, Pages 41-55
- Tan, B., Biglari-Abhari, M., Salcic, Z., (**DASIP - 2016**) *A System-level Security Approach for Heterogeneous MPSoCs*, Proceedings of IEEE Conference on Design & Architectures for Signal & Image Processing (DASIP-2016), Rennes, France, Oct. 2016
- Hemmati, M., Biglari-Abhari, M., Berber, S., & Niar, S. (**DSD - 2014***). HOG Feature Extractor Hardware Accelerator for Real-time Pedestrian Detection*. In 2014 17TH EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN (DSD) (pp. 543-550). Verona, Italy

Acknowledgement:

- **Dr. Benjamin Tan** **(part of this is based on his PhD research)**
  (currently postdoc researcher at New York University)
- **Dr. Maryam Hemmati** **(part of this is based on her PhD research)**
  (currently postdoc researcher at University of Auckland)

➢ **Prof. Smail Niar,** Université Polytechnique Hauts-de-France, Valenciennes (UPHF), Valenciennes, France
(collaboration on autonomous vehicles research)