Configurable Multi-protocol AUTOSAR-based Secure Communication Accelerator

Ahmed Hamed[1]; Mona Safar[2]; M. Watheq El-Kharashi[2]; Ashraf Salem[1]

[1]Mentor, A Siemens Business, Cairo, Egypt, [2]Ain Shams University, Cairo, Egypt

Abstract

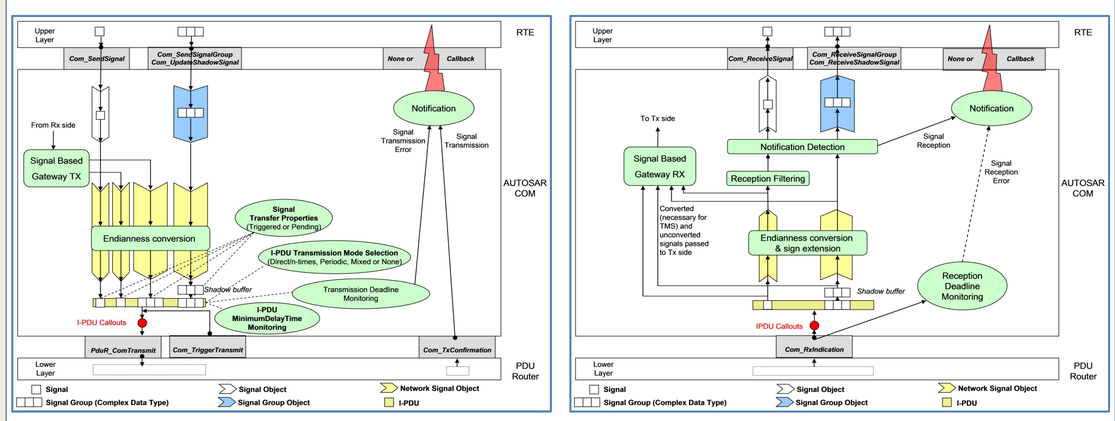
In this poster, an **AUTOSAR**-based secure communication accelerator is introduced. The aim of this accelerator is to cope with ongoing increase of exchanged information between Electronic Control Units (ECU) (e.g., **telematics** ECUs). The accelerator works with different bus widths (**CAN**, **Fr**, and etc.). It also supports long bus widths (e.g., **Ethernet**), which is used in different communication technologies (**V2V**, **V2X**, and etc.). These communication technologies deal with long information (**images**, **maps**, and etc.) that need to be transferred between ECUs securely in very short times. In this accelerator, set of instructions are designed to configure it and work with different network protocols simultaneously.

Motivations:

- **Configurable:**
  - The number of ECUs in modern cars has increased dramatically in the last years (for example, the number of ECUs in the **XC90** platform for **VOLVO** cars has increased from **38** ECUs in **2002** to **108** ECUs in **2015**.
  - Our accelerator can work with different ECUs (e.g., **Telematics** ECUs) and support different configurations.
- **Multi-protocol:**
  - The need to deal with other types of protocols (e.g., **Ethernet**), which are able to transfer long protocol data units and signals, has been aroused and it became possible to transfer other types of information (e.g., **images** and **maps**).
- **Secure:**
  - The **V2V** and the **V2X** concepts require the vehicle architecture to be secure.
- **Communication Accelerator:**
  - The number of the exchanged messages in **telematics** ECUs can exceed **600** messages.
  - The timing requirements of the new software applications (for example, some critical control messages in a car need to be processed and delivered within a very small delay such as **100 µs**).

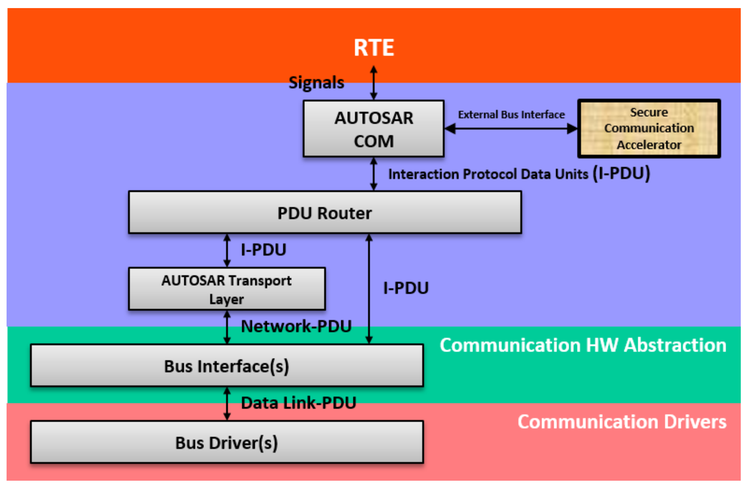
Introduction

The idea is to design a secure **AUTOSAR**-based communication accelerator for handling long protocol data units and signals by packing long transmitted signals into their corresponding protocol data units and unpacking long received signals from their corresponding protocol data units.



Figures 1,2: The AUTOSAR COM Module's Interaction Model for Transmission & Reception.

Then integrating the accelerated part (HW-based) with the core SW of the **AUTOSAR COM** module and measuring the performance improvement.
The exchange of the protocol data units between ECUs has to be done in a secure manner by using the appropriate cryptographic algorithms to prevent hackers from gaining access to the information inside these protocol data units and altering it.



Figures 3: Modified AUTOSAR Layered Software Architecture.

COM ISA Design

The proposed instruction format of the COM ISA is shown in TABLE 1.

TABLE 1: Proposed COM ISA Instruction Format.

B_{63} B_{56}	B_{55} B_{40}	B_{39} B_8	B_8 B_0
Opcode	Virtual Signal ID	Virtual Signal Value	RFU

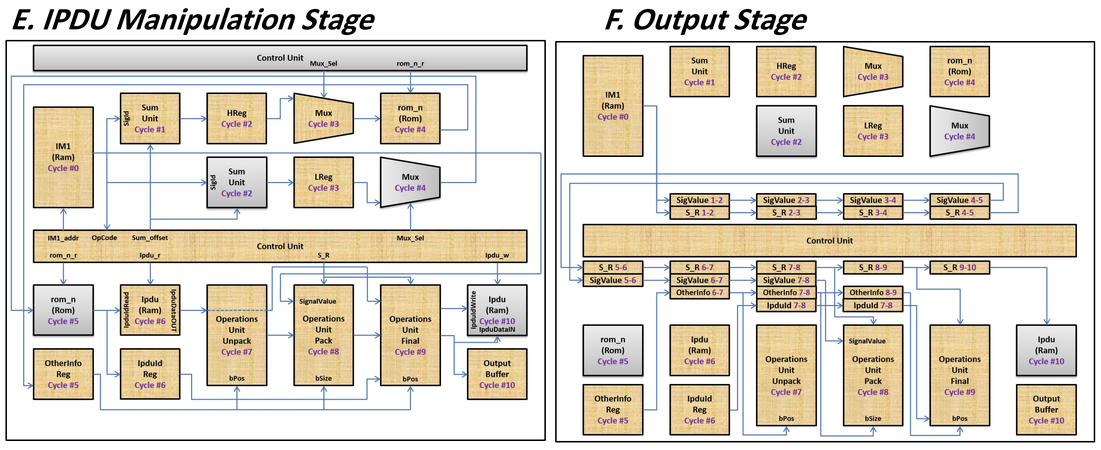
There are 4 supported instructions, as shown in TABLE 2.

TABLE 2: Supported Instructions.

Instruction		Op Code
Send Signal	Send Virtual Signal	x"FF"
Receive Signal	Receive Virtual Signal	x"11"
Send Last Virtual Signal in Pdu (new)		x"EE"
Receive Last Virtual Signal in Pdu (new)		x"22"

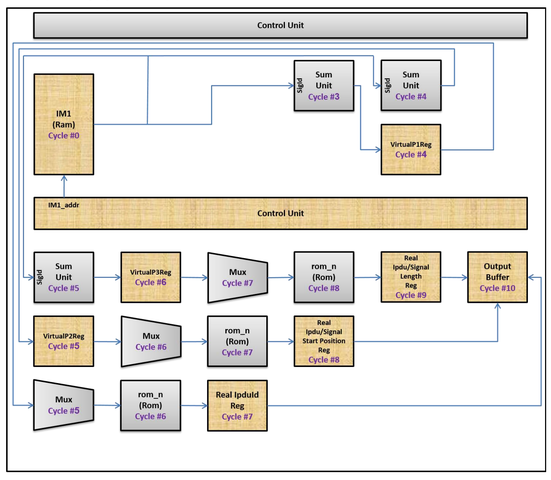
Each long ComSignal will be handled as a consecutive group of virtual ComSignals, where each virtual ComSignal will be handled as a normal UINT32 ComSignal. This design reuse the previously supported instructions (i.e., Send Signal and Receive Signal). To process the proposed COM ISA (for Send Signal and Receive Signal instructions), we designed a 6-stage pipelined processor. Each stage takes two hardware execution cycles. Thus, a new COM instruction is fed into the proposed processor each two cycles. The stages of the processor are shown below.

A. COM Instruction Fetch
B. COM Instruction Decode Stage
C. COMSignal Properties Fetch Stage
D. IPDU Fetch Stage
E. IPDU Manipulation Stage
F. Output Stage



Figures 4,5: The Non-Pipelined and the Pipelined Versions of the COM Processor.

We added two extra instructions (i.e., **Send Last Virtual Signal in Pdu** and **Receive Last Virtual Signal in Pdu**). These instructions will do the normal handling of the previously supported instructions (i.e., Send Signal and Receive Signal) in addition to communicating the properties of long ComIPDUs to the output buffer (i.e., **Real Ipdu Id**, **Real Ipdu/Signal Start Position**, and **Real Ipdu/Signal Length**). The changes in the COM processor to support the new instructions are shown in Figure 6.



Figures 6: The changes in the Non-Pipelined Version of the COM Processor.

The configured Virtual ComSignals and ComIPDUs properties are also encoded in a way that enables the proposed processor to execute the packing and unpacking operations appropriately. The proposed encoding is illustrated in TABLE 3.

TABLE 3: Encoding of Virtual ComSignal Configured Properties.

	B_{31-26}	B_{25-20}	B_{19-14}	B_{13-12}	B_{11-0}
Word 0	Init Value				
Word 1	Invalid Value				
Word 2	Virtual Ipdu Id = Virtual Ipdu Signal Start Position (in words "64-bits words")				
Word 3	bPos	UbPos	bSize	E	Not used
Word 4 (new)	Ipdu Signal Update Bit Position (in bytes)				
Word 5 (new)	Real Ipdu Id				
Word 6 (new)	Real Ipdu/Signal Start Position (in bytes)				
Word 7 (new)	Real Ipdu/Signal Length (in bytes)				