UNIT 2

EMBEDDED SYSTEMS MODELING AND DESIGN AND CPS

A cyber-physical system (CPS) is an integration of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa. As an intellectual challenge, CPS is about the intersection, not the union, of the physical and the cyber. It is not sufficient to separately understand the physical components and the computational components. We must instead understand their interaction.

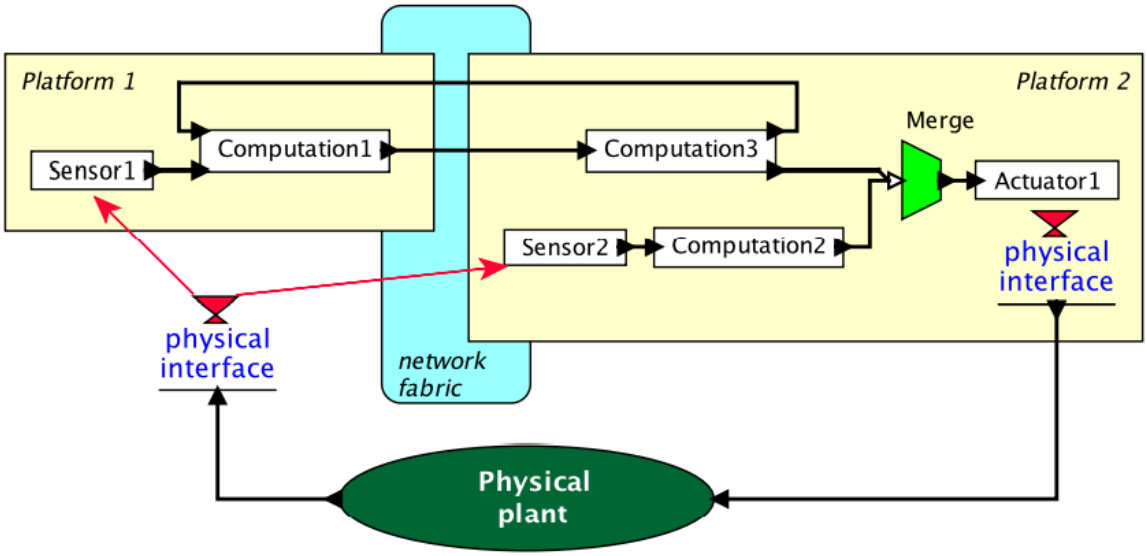


Figure 1 Example structure of a cyber-physical system.

There are considerable challenges in making such a system work. First, controlling the vehicle is not trivial. The main actuators are the four rotors, which produce a variable amount of downward thrust. By balancing the thrust from the four rotors, the vehicle can take off, land, turn, and even flip in the air. How do we determine what thrust to apply? Sophisticated control algorithms are required. Second, the weight of the vehicle is a major consideration. The heavier it is, the more stored energy it needs to carry, which of course makes it even heavier. The heavier it is, the more thrust it needs to fly, which implies bigger and more powerful

Figure 1.2: The STARMAC quadrotor aircraft in flight (reproduced with permission).

There are considerable challenges in making such a system work. First, controlling the vehicle is not trivial. The main actuators are the four rotors, which produce a variable amount of downward thrust. By balancing the thrust from the four rotors, the vehicle can take off, land, turn, and even flip in the air. How do we determine what thrust to apply? Sophisticated control algorithms are required.

Second, the weight of the vehicle is a major consideration. The heavier it is, the more stored energy it needs to carry, which of course makes it even heavier. The heavier it is, the more thrust it needs to fly, which implies bigger and more powerful motors and rotors. The design crosses a major threshold when the vehicle is heavy enough that the rotors become dangerous to humans. Even with a relatively light vehicle, safety is a considerable concern, and the system needs to be designed with fault handling.

Third, the vehicle needs to operate in a context, interacting with its environment. It might, for example, be under the continuous control of a watchful human who operates it by remote control. Or it might be expected to operate autonomously, to take off, perform some mission,

return, and land. Autonomous operation is enormously complex and challenging because it cannot benefit from the watchful human. Autonomous operation demands more sophisticated sensors. The vehicle needs to keep track of where it is (it needs to perform localization). It needs to sense obstacles, and it needs to know where the ground is. With good design, it is even possible for such vehicles to autonomously land on the pitching deck of a ship. The vehicle also needs to continuously monitor its own health, to detect malfunctions and react to them so as to contain the damage.

It is not hard to imagine many other applications that share features with the quadrotor problem. The problem of landing a quadrotor vehicle on the deck of a pitching ship is similar to the problem of operating on a beating heart (see Example 1.1). It requires detailed modeling of the dynamics of the environment (the ship, the heart), and a clear understanding of the interaction between the dynamics of the embedded system (the quadrotor, the robot).

The Design Process

Figure 1.3 shows the three major parts of the process, modeling, design, and analysis. Modeling is the process of gaining a deeper understanding of a system through imitation. Models imitate the system and reflect properties of the system. Models specify what a system does. Design is the structured creation of artifacts. It specifies how a system does what it does. Analysis is the process of gaining a deeper understanding of a system through dissection. It specifies why a system does what it does (or fails to do what a model says it should do). As suggested in Figure 1.3, these three parts of the process overlap, and the design process iteratively moves among the three parts. Normally, the process will begin with modeling, where the goal is to understand the problem and to develop solution strategies.

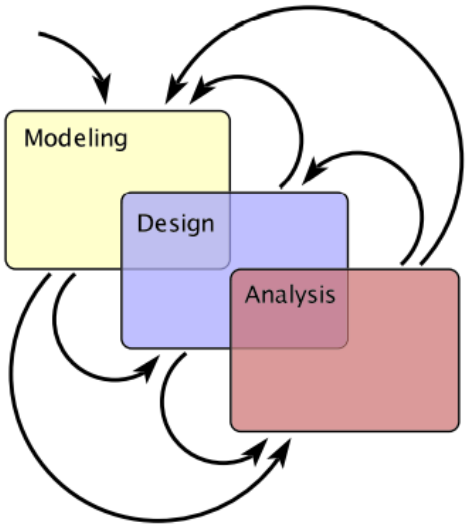
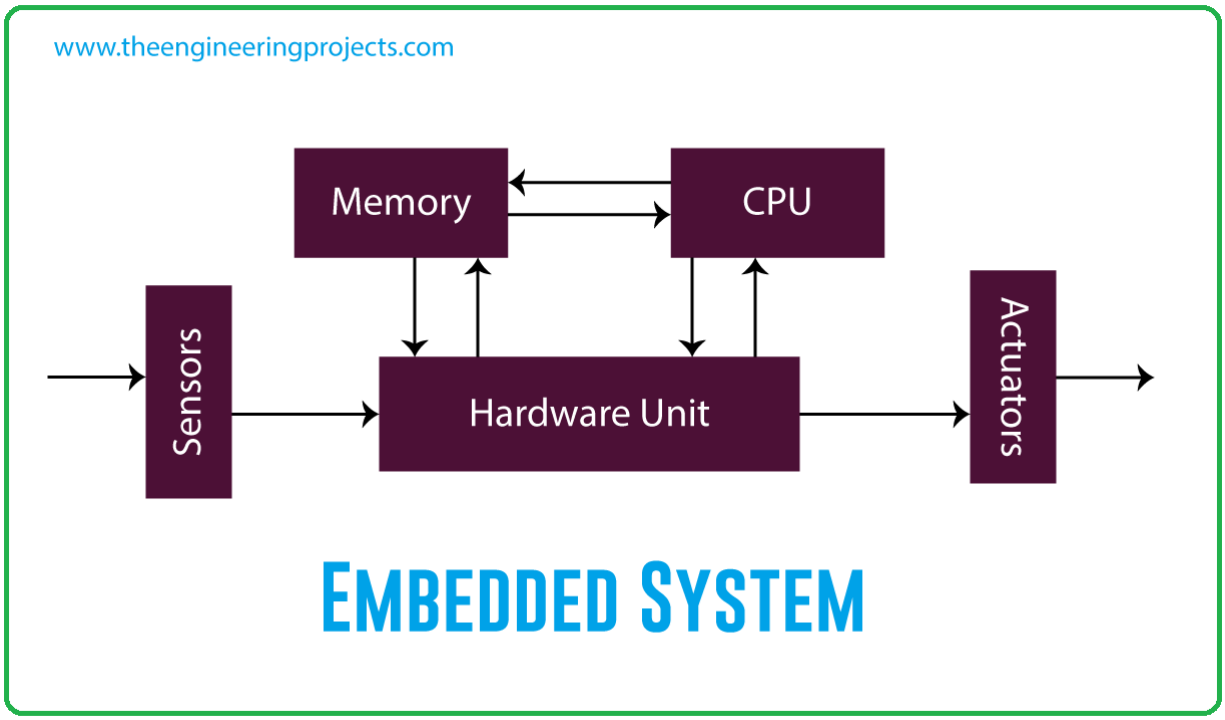


Figure 1.3: Creating embedded systems requires an iterative process of modeling, design, and analysis.

An embedded system is a combination of computer hardware and software designed for a specific function. Embedded systems may also function within a larger system. The systems can be programmable or have a fixed functionality. Industrial machines, consumer electronics, agricultural and processing industry devices, automobiles, medical equipment, cameras, digital watches, household appliances, airplanes, vending machines and toys, as well as mobile devices, are possible locations for an embedded system.

While embedded systems are computing systems, they can range from having no user interface (UI) -- for example, on devices designed to perform a single task -- to complex graphical user interfaces (GUIs), such as in mobile devices. User interfaces can include buttons, LEDs (light-emitting diodes) and touchscreen sensing. Some systems use remote user interfaces as well.



2.1 Platform components

Three components of an embedded system

An embedded system has mainly three components

1. hardware,
2. application specific software, and
3. real-time operating system

Other Components of the Embedded System

- Power supply. For the embedded system the power supply is the key component to provide the power to the embedded system circuit. ...
- Processor. ...
- Memory. ...
- Timers counters. ...
- Communication ports. ...

- Output and Input. ...
- Circuits used in application.

Embedded systems are found in a variety of common electronic devices, such as:

(a)consumer electronics -- cell phones, pagers, digital cameras, camcorders, videocassette recorders, portable video games, calculators, and personal digital assistants; (b) home appliances -- microwave ovens, answering machines, thermostat, home security, washing machines, and lighting systems;

(c) office automation -- fax machines, copiers, printers, and scanners;

(d) business equipment -- cash registers, curbside check-in, alarm systems, card readers, product scanners, and automated teller machines;

(e) automobiles --transmission control, cruise control, fuel injection, anti-lock brakes, and active suspension

2.2 - Embedded Systems definition, specification, and languages.

An embedded system is nearly any computing system other than a desktop computer. An embedded system is a dedicated system which performs the desired function upon power up, repeatedly.

Embedded systems are found in a variety of common electronic devices such as consumer electronics ex. Cell phones, pagers, digital cameras, VCD players, portable Video games, calculators, etc.

Structure-oriented models describe the system's physical modules and the interconnections between them. They are well-suited at describing a particular architecture, such as a four-processor implementation with shared memory and an eight-processor implementation with cross-bar communication.

The major areas of the design process are

I. Ensuring a sound software and hardware specification.
II. Formulating the architecture for the system to be designed.

III. Partitioning the h/w and s/w.
IV. Providing an iterative approach to the design of h/w and s/w.

The important steps in developing an embedded system are

I. Requirement definition.
II. System specification.
III. Functional design
IV. Architectural design
V. Prototyping.

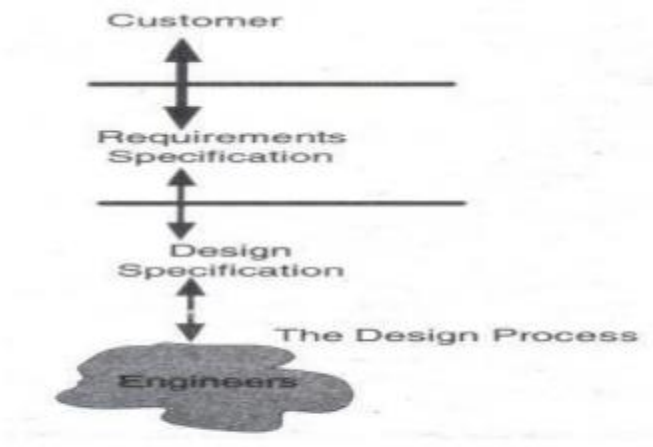
The system design specification

The System Design Specification (SDS) is a complete document that contains all of the information needed to develop the system. Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. System design specification serves as a bridges between the customers and designers as shown in figure 4.5. Figure 4.5: The Customer, the requirement, the design and the engineer The requirement specifications provides a view from the outside of the system, design specification provides a view from the inside looking out as well. Design specification has 2 masters:

It must specify the system‘s public interface from inside the system. It must specify how the requirements defined for and by the public interface are to be met by the initial functions of the system.

Five areas should be considered are:

I. Geographical constraints.
II. Characterization of and constraints on interface signals.
III. User interface requirements Temporal constraints.
IV. Electrical infrastructure consideration
V. Safety and reliability



The Customer, the requirement, the design and the engineer

Best Embedded Systems Programming Languages

Embedded Systems Programming is an exclusive industry and only a few programming languages are allowed entry because there are specific requirements such as low usage of resources as well as low-level system access.

Embedded systems are controllers that are handled using real-time operating systems. They are embedded in different devices such as smartphones, watches, cars, etc. to make them "smarter".They are programmed using embedded systems programming that is quite different than traditional programming as it requires low-level system access and as low usage of resources as possible.

1. C

C has multiple features like low-level access to memory, a loose data typing policy, ease of porting embedded programs, etc. It is also very fast as compared to other similar programming languages which only helps its case!!!

- The loose data typing policy of C makes it quite suitable.

- It is simple to port the embedded programs from one device to the next as compared to other languages.
- The widespread community in C provides vast support for Embedded Systems Programming.

2. C++

C++ is less popular than C in regards to embedded systems but the addition of object-oriented programming. It also has low-level access to memory like C that makes it quite suitable.

- C++ is more secure than C because of its use of string literals, enumeration constants, templates etc.
- Overloaded functions and constructors in C++ are an asset for embedded systems programming.
- The object oriented nature of C++ is also quite useful for complex embedded systems programming.

3. Java

Java embedded system programs can be ported onto different platforms because of its WORA("write once, run anywhere") functionality. Also, Java is quite popular and widely used in embedded systems programming because of the various DevOps tools available.

- Java can be used to write extensible, portable, and downloadable embedded systems applications.
- There are many DevOps tools and libraries in Java that make it suitable for Embedded Systems Programming.
- The Java Virtual Machine ensures that embedded systems programmed in Java are portable and can be used for different IoT platforms.

4. Python

Python is popular language as compared to C or C++. While it is currently used in **only about 5% of all embedded system code**

- Python is a popular language and known for its writability, concise, readable coding style, and error deduction.
- Python is much handier in the case of complicated embedded systems such as those using neural networks.
- Real time embedded systems use Python quite often. *MicroPython* is a good example of a lean and efficient implementation for this.

5. Rust

Rust also has a similar syntax to C++ and so can be easily integrated into existing C/C++ code bases. Rust is good in embedded systems programming because of high performance, multiple safety features, a typestate programming style as well as zero cost abstractions which makes it ideal for embedded systems programming.

- Rust allows memory management using both dynamic and static methods using various tools.
- Rust can be used to program a wide range of embedded systems from small micro controllers to large multifaceted systems.
- There is a large community support in Rust for embedded systems programming.

6. Ada

Ada can be called a patriotic language as it was specifically designed by the US Department of Defense for real-time embedded systems. Ada also has many safety-critical support features that make it useful in military applications, avionics, space systems, etc.

- Ada is useful for embedded systems programming because of strong typing, run-time checking, parallel processing, exception handling, generics etc.
- Ada packages can be compiled separately as it was created for the development of large software systems.
- Ada is used in critical systems as it supports run-time checks for bugs such as unallocated memory range violations, off-by-one errors, array access errors etc.

7. Lua

Lua uses a clear syntax and as low memory usage as possible. Other functionalities that are quite useful are garbage collection, coercion, closures, proper tail calls, etc.

- The base language in Lua is quite light as it has various meta-features that can be extended as required.
- Lua can implement object oriented programming using first-class functions and tables.
- Lua is cross-platform and it supports a C API that can be embedded into applications.

8. B#

B# is excellent in Embedded Systems programming as it was designed primarily for it. B# has a tiny core and as low a memory footprint as possible (which certainly cannot be said about Ranvir!!!). B# also supports the object-oriented paradigm with classes, interfaces, handlers, high-level mapping, etc.

- B# is well suited for small scale embedded systems programming because of its tiny core and small memory constraints.
- B# is type safe and does not allow any pointer manipulation which is an asset in embedded systems programming.
- The code written in B# is directly mapped to a tight instruction set. This reduces the runtime in low resource embedded devices.

9. Verilog

Verilog is is a shortened name for "**VERIfication of LOGic**" Verilog is also a Hardware Description Language(HDL) and it is used frequently in embedded systems programming as it provides low-level access to the system hardware.

- Verilog can be used to design custom hardware that is required by the embedded system.
- Verilog has a module hierarchy wherein the modules can communicate with each other using the input, output, and bidirectional ports.
- The modules contain concurrent and sequential statement blocks. These blocks are executed concurrently and so Verilog is a dataflow language.

10. Embedded C++

Embedded C++ is a subpart of C++ . However, Embedded C++ was created specifically for embedded systems programming and thus it contains the useful features of C++ and omits other features such as multiple inheritance, namespaces, templates, etc.

- Embedded C++ aims to minimize the code size and maximize the execution efficiency in regards to embedded systems programming.
- A standard C++ compiler can be used to compile the embedded systems programming done in Embedded C++.
- Some of the specific compilers for Embedded C++ are provided by Freescale Semiconductor, Green Hills Software etc.

2.3 Concepts, requirements, examples

Requirements of an embedded system

The requirements of embedded systems are different from the requirements of the traditional computer based system. To develop and adopt embedded systems, software such as C++, C, ADA,

etc, are used and some systems which have to specialised use operating systems such as Linux, OSE, Nucleus RTOS, Windows CE, and ThreadX.

Usually, with application development, there is a consideration to external factors such as temperature and other environmental factors which may affect performance, however with embedded software development this is not the case.

Embedded Systems Examples

There are many things with embedded systems incorporated in the Internet of Things (IoT), as well as in machine to machine (M2M) devices. Exceptionally versatile and adaptable, embedded systems can be found in all smart devices today. It is difficult to find a single portion of modern life that doesn't involve this technology. Here are some of the real-life examples of embedded system applications.

- Central heating systems
- GPS systems
- Fitness trackers
- Medical devices
- Automotive systems
- Transit and fare collection
- ATMs
- Factory robots
- Electric vehicle charging stations
- Interactive kiosks

1. Central Heating Systems

Central heating systems convert the chemical energy into thermal energy in a furnace room and transfer that energy into heat, which is then delivered to numerous spaces within a building. It is important for these systems to have thermostat controls to adjust the temperature, which is achieved by an embedded system.

If a central heating system isn't provided with temperature controls, it can lead to overheating one room while leaving another room cold. The right thermostat controls will allow you to adjust the temperature to a comfortable level and save energy extensively.

Embedded system examples in central heating can be found in a range of structures that require temperature control, both for comfort and for management of temperature-sensitive goods.

Examples include:

- Office buildings
- Factories
- Grocery stores
- Homes
- Schools
- Hospitals

2. GPS Systems

The GPS is a navigation system that uses satellites and receivers to synchronize data related to location, time, and velocity. The receiver or device that receives the data has an integrated embedded system to facilitate the application of a global positioning system. The embedded GPS devices allow people to find their current locations and destinations easily. Thus, they are gaining rapid momentum and becoming the most widely used navigation tools for automobiles.

Nowadays, GPS systems are generally used in:

- Cars
- Mobile devices
- Palmtop

3. Fitness Trackers

Fitness trackers are wearable devices that can monitor your health and track activities like sleeping, running, and walking. These devices use embedded systems to garner data related to your heart rate, body temperature, and the number of footsteps, which is further sent to servers via WAN like LTE or GPRS.

Fitness trackers are generally used for:

- Monitoring personal activity
- Medical monitoring
- Sports training

4. Medical Devices

Medical devices in healthcare facilities have been incorporating embedded systems for quite some time. A new class of medical devices use embedded systems to help treat patients who need frequent monitoring and constant attention at home. These systems are embedded with sensors to gather data related to patients' health like heart rate, pulse rate, or readings from implants, which are sent to a cloud where a doctor can review patient data on their device wirelessly. Medical devices have been widely used for diagnosing and treating patients efficiently, and some of their examples are:

- Pacemaker
- Defibrillator
- Ultrasound scanners

5. Automotive Systems

Automotive embedded systems are designed and installed to enhance the safety of automobiles. Thanks to the safety systems in vehicles, the traffic fatality rate has plummeted in recent years. Automobile industries are going the extra mile to reinforce automobiles with advanced technology systems and sensors, which is not possible without embedded systems.

Some key examples of an active safety system include adaptive speed control, car breakdown warning, pedestrian recognition, merging assistance, airbags, and more. These are a few features anticipated to mitigate the risk of accidents and foster the demand for embedded systems across the globe.

Some more examples of automotive embedded systems include:

- Car navigation system
- Anti-lock braking system
- Vehicle entertainment system

6. Transit and Fare Collection

Automated Fare Collection (AFC) is a ticketing system that allows passengers to pay the fare through ticket vending machines or online services. These systems were originated with coins and tokens but have been replaced with magnetic stripe cards or smart cards. An AFC is a basic station device comprising a ticket vending machine, automatic gate machine, and ticket checking machine. These components are embedded systems that ensure faster transactions, seamless operations, and more efficient payment collection.

While city transit bus and commuter rails still use paper tickets and passes, urban transit systems have adopted AFC with smart cards, which are inexpensive technologies and offer additional security along with data collection options.

Automated fare collection systems are generally found at:

- Metro stations
- Bus stations
- Railway stations

If you are looking for embedded processor examples in the transportation sector, see some of our customer stories, sharing how Digi embedded System-on-Modules are designed into transit and vehicle applications:

- EMtest

- TransData

7. ATMs

An automated teller machine (ATM) is a computerized machine used in banking that communicates with a host bank computer over a network. The bank computer verifies all the data entered by the users and stores all transactions, while the embedded system in the ATM displays the transaction data and processes inputs from the ATM keyboard.

An ATM is mostly used to:

- Withdraw cash
- Check account balance and transactions details
- Deposit money into another account

8. Factory Robots

Factory robots are designed to perform high-precision tasks under dangerous work conditions. They have an integrated embedded system to connect different subsystems. In a typical mechanical job, robots employ actuators, sensors, and software to perceive the environment and derive intended output safely.

Without an embedded system, robots would have to rely on external control or computing systems. This, in turn, can elevate the safety risks due to delay or failure in the connection link between the factory robot and its external computing system. Today, as Industry 4.0 comes to fruition, these systems are integrating artificial intelligence and machine learning to make equipment smarter, safer and more effective — for example, enabling machines to identify defects that the human eye wouldn't see, and remove these from production.

Factory robots have a range of applications:

- Assembly line
- Quality monitoring
- Welding
- Painting
- Palletizing

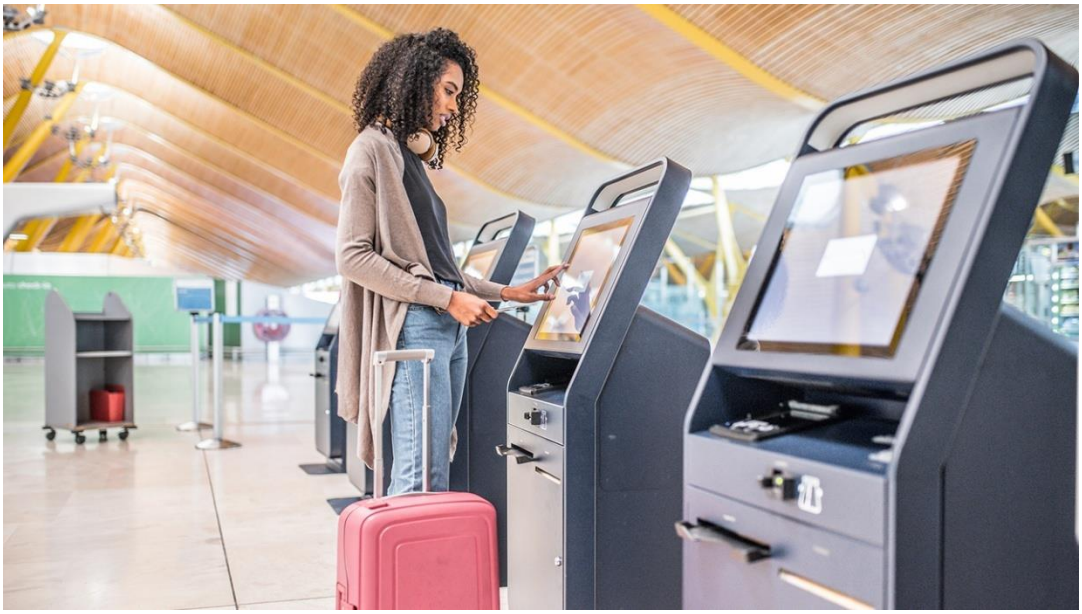
9. Electric Vehicle Charging Stations

Electric vehicle charging stations are equipped with charging points or units that supply electric power to charge connected vehicles. An embedded system resides in the charging station to provide processing for graphics displays, report any issues with the device and alert technicians when maintenance is required. This embedded solution provides an easy and cost-effective approach to monitoring and maintaining the charging infrastructure. A number of Digi customers, such as AddÉnergie, are developing solutions to serve this growing market.

Some of the common uses of electric vehicle charging stations include:

- Charging vehicles
- Swapping batteries
- Parking vehicles

10. Interactive Kiosks



Self-service kiosks are designed to offer services and information to end-users in environments where human employee presence isn't possible or cost-effective. For instance, these machines and terminals allow a passenger in an empty airport to buy a meal at 4 am without interacting with human workers. Interactive kiosks come in all shapes and sizes, from simple coffee dispensing systems to complex vending machines and fuel stations with high-definition graphics. For this reason, it is important for embedded developers to work with a scalable product line like Digi ConnectCore® 8X/8M system-on-modules (SOMs), which support development of product lines with scaling levels of functionality.

An embedded system provides the processing for connected, self-service kiosk machines, offering an interactive consumer experience. These systems can be developed to function in remote and outdoor environments and deliver information and services even in extreme weather conditions. They can also eliminate downtime for real-time applications and have expandable I/O options designed for workload consolidation.

2.4 Embedded system models at different abstraction levels.

Levels of abstraction

A level of abstraction is usually understood in terms of the language of design. Often, the

language defines the level, but that doesn't have to be the case. A level of abstraction is determined by the objects that can be manipulated and the operations that can be performed on them. In programming terms, the objects are data types and the operations are the operators that can be used in expressions and control constructs.

Interpretation

Interpretation is a technique for dynamically realizing the semantics of a language. A good example of a level of abstraction being implemented by interpretation is a micro-coded processor. The instructions visible to the programmer are implemented by a lower level processor which interprets the programmer-visible instructions. You can also make the argument that the use of a low-level VLIW processor to implement an algorithm is an example of interpretation. There aren't many other examples of the use of interpretation in hardware design.

Translation

Translation is the most easily recognized form of abstraction implementation. Synopsys' Design Compiler translates RTL Verilog into a netlist, a process called logic synthesis. There has never been a widely accepted definition of the register-transfer level. The traditional definition is that RTL is whatever Design Compiler accepts. A better definition is that RTL is a representation where the activity in each clock cycle is explicitly defined.

Behavioral synthesis was an easily identified analog of logic synthesis. It translated language at a behavioral level to RTL (or directly to gates). In general, behavioral level code is written using the same data types as RTL, but it is unscheduled and unallocated. That is, the data objects and the operations on them are specified, but the resources and the schedule of cycles required are not specified.

Extension

Extension is not as widely recognized in hardware design as translation and interpretation, but is potentially the most useful. Extension is accomplished by defining new data types and operations in the same language that is used in the next lower level. Object oriented languages make creating new levels of abstraction in a single language particularly easy, and in fact, that

is the primary virtue of C++. SystemC is nothing more than a hardware level of abstraction implemented by extension in C++. More important is that using C++, additional data types and operations can be defined to create higher levels than the base SystemC level.

2.5 Test benches,

As a general rule, for each component you develop, there should be at least one test bench.

Different types of Test Benches

1. Testing a Single Architecture Component
2. Multiple Architectures
3. Automatic Testing with Assert
4. Testing with Simulated Timing
5. Timing Checks
6. Modelling and Testing Synchronous Systems

Testing a Single Architecture Component

Defining a Component

When you create a component, you specify the **entity** and at least one **architecture**.

- The entity defines the inputs and output signals.
- The entity describes the logical relationship between the input and output signals.

You can think of an entity as the description of a device, as viewed from the outside. For example:

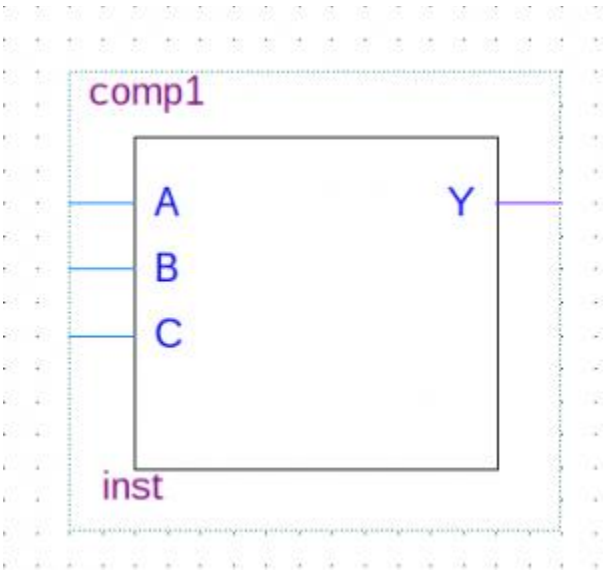
```
library ieee;
use ieee.std_logic_1164.all;


-- Component 1 - very simple device that performs a logical OR on three single bit inputs
entity comp1 is
 port( A: in std_logic;
```

```
 B: in std_logic;
 C: in std_logic;
 Y: out std_logic
);
end comp1;
```

The highlighted line is the start of the entity declaration.

- The name of the entity is **comp1**
- This entity has a single architecture **inst1**

We can represent this component visually as follows:



Component showing input signals A,B and C with an output signal Y

Below the entity is the architecture that contains the logic

```
--This component only has one architecture
architecture inst1 of comp1 is
begin
 Y <= A or B or C;
end inst1;
```

We could use this component in a design, synthesise it and test on real hardware. However, it is advisable (where possible) to simulate it first. Tools included with Altera Quartus II allow us to do this using either:

- Vector Waveform File (simulation)
- SignalTap on Hardware
- Running a “Test Bench” in ModelSim (simulation)

http://blogs.plymouth.ac.uk/embedded-systems/fpga-and-vhdl/test-benches/part-1-single-architecture-test/

2.6 Design under test

Embedded Testing

Embedded Testing is a testing process for checking functional and non-functional attributes of both software and hardware in an embedded system and ensuring that the final product is defect free. The main purpose of Embedded testing is to verify and validate whether the final product of embedded hardware and software fulfill the requirements of the client or not.
Embedded Software testing checks and ensure the concerned software is of good quality and complies with all the requirements it should meet. Embedded software testing is an excellent approach to guarantee security in critical applications like medical equipment, railways, aviation, vehicle industry, etc. Strict and careful testing is crucial to grant software certification.

How to perform Embedded Software Testing

In general, you test for four reasons:

- To find bugs in software
- Helps to reduce risk to both users and the company
- Cut down development and maintenance costs
- To improve performance

In Embedded Testing, the following activities are performed:

1. The software is provided with some inputs.

2. A Piece of the software is executed.

3. The software state is observed, and the outputs are checked for expected properties like whether the output matches the expected outcome, conformance to the requirements and absence of system crashes.

Difference: Embedded testing and Software Testing

Software Testing	Embedded Testing
Software testing is related to software only.	Embedded testing is related to both software as well as hardware.
On average 90% testing done in the world is purely manual black box testing.	Embedded testing is done on embedded systems or chips it can be a black box or white box testing.
Primary areas of testing are GUI checks, functionality, validation and some level of database testing.	Primary areas of testing are the behavior of the hardware for the no. of inputs given to it.
Software testing is majorly performed on client-server, web and mobile based applications.	Embedded testing generally performed on the Hardware.
e.g., Google Mail, Yahoo Mail, Android applications.	e.g., Machines of healthcare domain, Microcontrollers used in computers.

Embedded Software Testing

Some of the challenges that one can face during Embedded software testing:

Hardware Dependency

Hardware dependency is among the main difficulties faced during embedded software testing because of limited access to hardware. However, Emulators and Simulators may not precisely represent the behavior of the actual device and could give a wrong sense of system performance and application's usability.

Open Source Software

The majority of the embedded software components are open source in nature, not created in-house and absence of complete test available for it. There is a wide range of test combinations and resulting scenarios.

Software vs. Hardware Defects

Another aspect is when software is being developed for a freshly created hardware, during this process high ratio of hardware defects can be identified. The found defect is just not limited to software. It may be related to hardware also.

Reproducible Defects

Defects are harder to reproduce/recreate in the case of the embedded system. That enforces the embedded testing procedure to value every defect occurrence substantially higher than in a standard case, other than to gather as much data as could sensibly be required to alter the system to find the foundation of the defect.

Continuous Software Updates

Embedded systems require regular software updates like the kernel upgrade, security fixes, different device drivers, etc. Constraints identified with the software updates influence makes bug identification difficult. Additionally, it increases the significance of build and deployment procedure.

2.7 Intellectual Property components.

Embedded systems developers want to protect their intellectual property(IP) for financial, competitive, and legal reasons. Aside from protecting their own IP, developers may be obligated to secure libraries or rights managed content obtained from an outside vendor. Failing to protect the IP of outside vendors may expose developers to legal and financial risks.

These techniques rely on software, hardware, or a combination of both to prevent reverse engineering and IP theft.

Software-based IP Protection

Software methods for protecting IP in embedded processors are similar to those employed in the broader software industry: anti-disassembly, anti-debug, and tamper resistance.

Anti-disassembly. Anti-disassembly techniques attempt to make the static view of an executable reveal as little information as possible about the softwareIP in the executable. When employed correctly, anti-disassembly techniques make it more challenging for an adversary to understand, andthus to reverse-engineer, the software.

An example of an anti-disassembly technique is self-modifying code in which a software program modifies itself at run time. An adversary viewing the software statically will have difficulty figuring out thecode that actually executes at run time. Self-modifying code is interesting, but implementing it without complicating the development process is a challenge.

Anti-debug. Anti-debug refers totechniques that prevent successful debug of software. A good anti-debugmechanism is one that doesn't alert the attacker to its existence. Ifattackers can't detect the presence of anti-debug mechanisms in thesoftware they are trying to reverse-engineer, they will be less likelyto succeed in identifying and disabling these mechanisms.

Anti-debug consists of techniques for debug detection and forresponse to debug attempts. Debug detection techniques includemonitoring changes in the values of debug registers, self-scanning forbreakpoints or emulation interrupts, and/or measuring time delaybetween various points in the code execution.

Once a process detects that it is being debugged, it may respond ina manner that prevents IP theft by, for example, avoiding itsproprietary algorithm and instead executing a more conventionalalgorithm.

Tamper resistance. Tamperresistance mechanisms are introduced into a software program in orderto prevent adversaries from modifying the program. Adversaries modifysoftware programs in order to make it behave in a manner different fromthat intended.

For example, to prevent software cloning ” making unauthorizedcopies of software ” a program may read and verify the unique device IDas a pre-requisite for execution. If the ID read does not match thatstored in the program, the program halts itself because there is a highprobability that it has been cloned and placed into a different devicethat has a different unique ID.

An adversary cloning the software program may attempt to patch it ina manner that makes it bypass the unique device ID check.Anti-tampering techniques hamper malicious code modifications by makingthe code more difficult to understand and/or by making the code likelyto fail in case of tampering.

Making the code more difficult to understand may be achieved throughthe use of a code obfuscator. Code obfuscators are essentiallycompilers that perform transformations on an input program in order toproduce a functionally equivalent but more difficult toreverse-engineer output program.

Making the code likely to fail in case of tampering may be achievedby calculating a hash of the software program at different pointsduring execution. If the program was tampered with, the calculated hashvalue will not equal the expected value, and the program may take stepsto evade the attacker.

In the absence of implementing a system with IP protection featuresintegral to its design, strategies that intelligently combineanti-disassembly, anti-debug, and tamper resistance can provide atleast some level of protection in inherently open systems.

Hardware-based IP Protection

Hardware methods for protecting IP in embedded processors attempt todrive up the cost of probing and reverse-engineering the system. Thesemethods can be implemented at the system level, board level or chiplevel.

System-level protection. One example of system-level protection is a mutual watchdog scheme.A small microcontroller and processor provide mutual watchdog timerservices. If either processor is halted or stops responding to theperiodic interprocessor communication (i.e. halted via JTAG, removedfrom board, etc), the other processor performs a countermeasure such aserasure of on-board FLASH contents, removal of power to the otherprocessor or in the most extreme case, physical destruction of theprocessors and memory devices themselves.

Board-level protection . Board-level protection methods can include methods which rely onobfuscating the hardware access points into the processors and datalines of memory devices. Board layout techniques and package selectioncan be employed to make it difficult to access critical data and signallines.

If sensitive information is transferred over unsecured data linesbetween memory devices and processors, these data lines can be probedand the IP can be intercepted during booting or other memory transfers.If critical signals cannot be accessed, they cannot be probed ortampered with.

Board layout can be done using blind vias and solid power and groundplanes to shield access to signal traces on inner layers. Critical datalines between FLASH and processor, as well as JTAG signals, can beplaced on inner copper layers.

Selecting BGA packages with fine ball pitch and placing criticalsignals away from the outermost rows of balls can further complicatemalicious attempts to probe signals carrying data between processorsand memory chips such as FLASH or DRAM.

Placing processor and memory packages as close together as possibleon the board makes it harder for attackers to locate and probe criticalsignals. These techniques make it difficult for attackers to accesssignals even when drilling through the PCB. The downside is that BGApackage selections and these board layout techniques (which add layersto the boards) will increase the cost of board manufacturing.

2.8 Discrete event simulation, semantics, algorithm

Discrete event simulation

For example, **a truck arrives at a warehouse, goes to an unloading gate, unloads, and then departs**. To simulate this, discrete event simulation is often chosen. Using discrete event simulation modeling, the movement of a train from point A to point B is modeled with two events, namely a departure and an arrival.

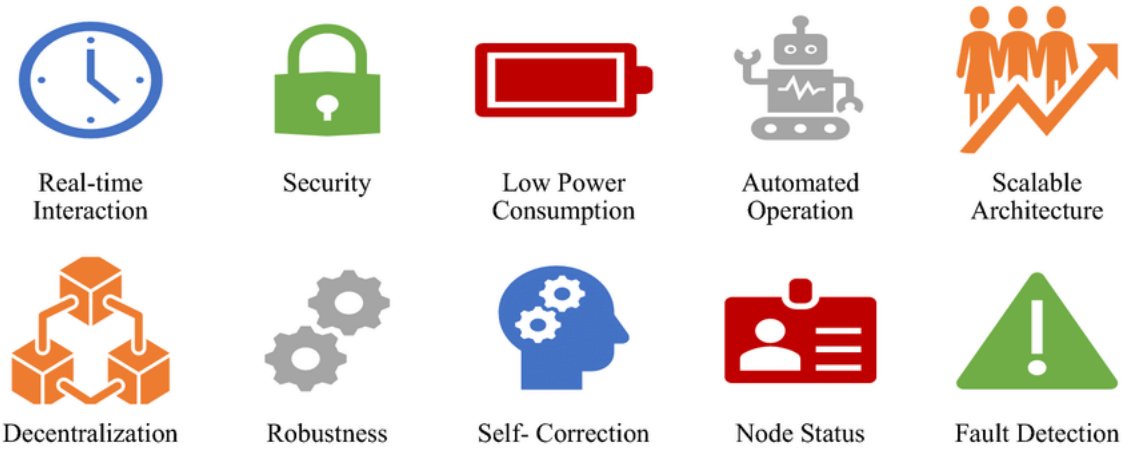
Discrete event simulation (DES) is a method of simulating the behaviour and performance of a real-life process, facility or system. DES is being used increasingly in health-care

services[24–26] and the increasing speed and memory of computers has allowed the technique to be applied to problems of increasing size and complexity. DES models the system as a series of 'events' [e.g. a birth, a stay in an intensive care unit (ICU), a transfer or a discharge] that occur over time. DES assumes no change in the system between events. In DES, patients are modelled as independent entities each of which can be given associated attribute information. In the case of neonatal simulation this may include parameters such as gestational age or weight at birth, hospital of birth, singleton/twin and current location. The information may be modified as time runs in the simulation model (e.g. the location will be changed depending on the status of the units in the network, and the level of care being received will be modified as the infant progresses). The simulation also accounts for resources. In the neonatal model the key resources are cots (with the highest level of care for each cot specified) and nurses. In order to care for an infant a unit must have the necessary cot and the necessary nursing staff (applying appropriate guidelines). The model allows each unit to work to a specified level of overcapacity regarding nursing, but will monitor the time each unit is undergoing overcapacity. DES models also allow for complex rules specifying where infants may be accepted; for example, there may be two ICUs, but with different facilities (e.g. surgery) or with different limits on gestational ages. DES thus allows complex decision logic to be incorporated that is not as readily possible in other types of modelling.

Simulation allows many 'what if?' scenarios to be tested. This allows decision-makers to test and better understand alternative ways in which a new policy may be best met.

2.9 DESIGN, AND ANALYSIS TECHNIQUES FOR DECENTRALIZED COMPUTER ARCHITECTURES, COMMUNICATION, AND HARDWARE-SOFTWARE SYSTEMS.

Cyber-Physical Production Systems (CPPSs) are complex manufacturing systems which aim to integrate and synchronize machine world and manufacturing facility to the cyber computational space. However, having intensive interconnectivity and a computational platform is crucial for real-world implementation of CPPSs.



Features of Decentralized Cyber Physical Systems

Decentralized System Architecture

The system architecture for Cyber Physical Production Systems (CPPS) consists of three distinct phases:

1) ***Automatic Production Plan Generation***: We automatically generate the production plan based on a system description and formalized specifications.

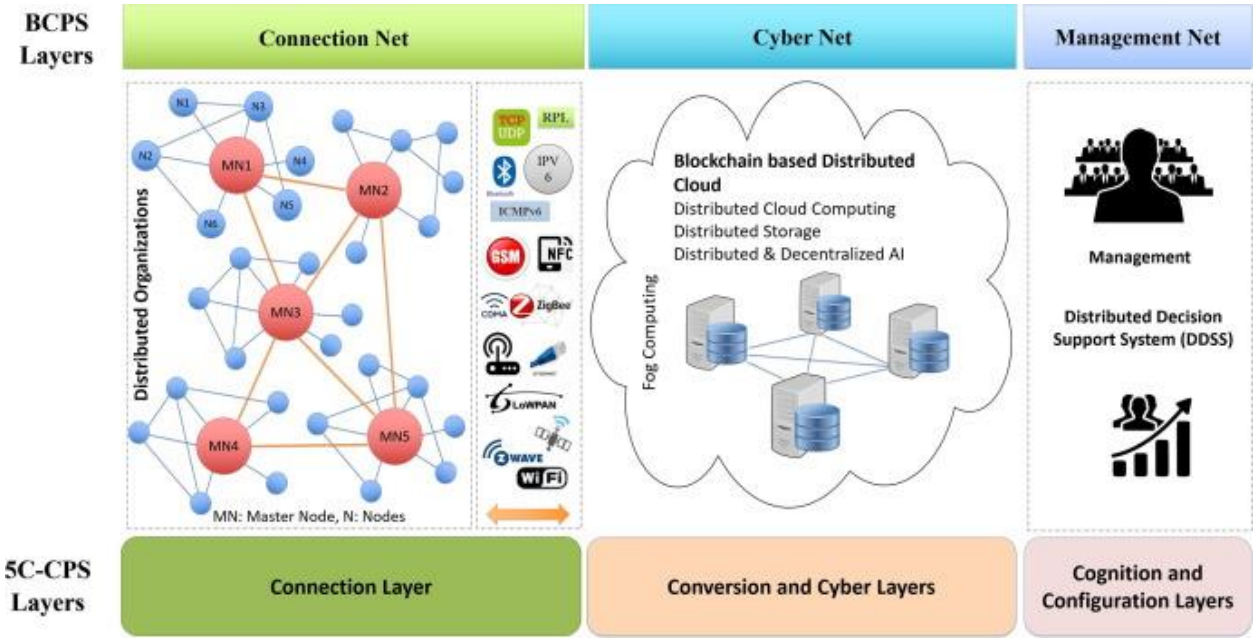
2) ***Production Plan Validation***: The timing, safety, and functional correctness of the generated production plan are validated.

3) ***Decentralized Two-Stage Consensus***: The decision-making process is built on a two-stage consensus algorithm, featuring a majority agreement on the safety and optimality and a unanimous agreement of all executing devices on the feasibility and authenticity of the plan. The hardware design of the architecture is visualized in Figure 2. We consider a mesh network of heterogeneous devices with varying functionalities and computational capabilities. The generation of the production plans is outsourced to cloud devices, whereas the functional validation can be performed on the computationally stronger devices in the mesh network as well.

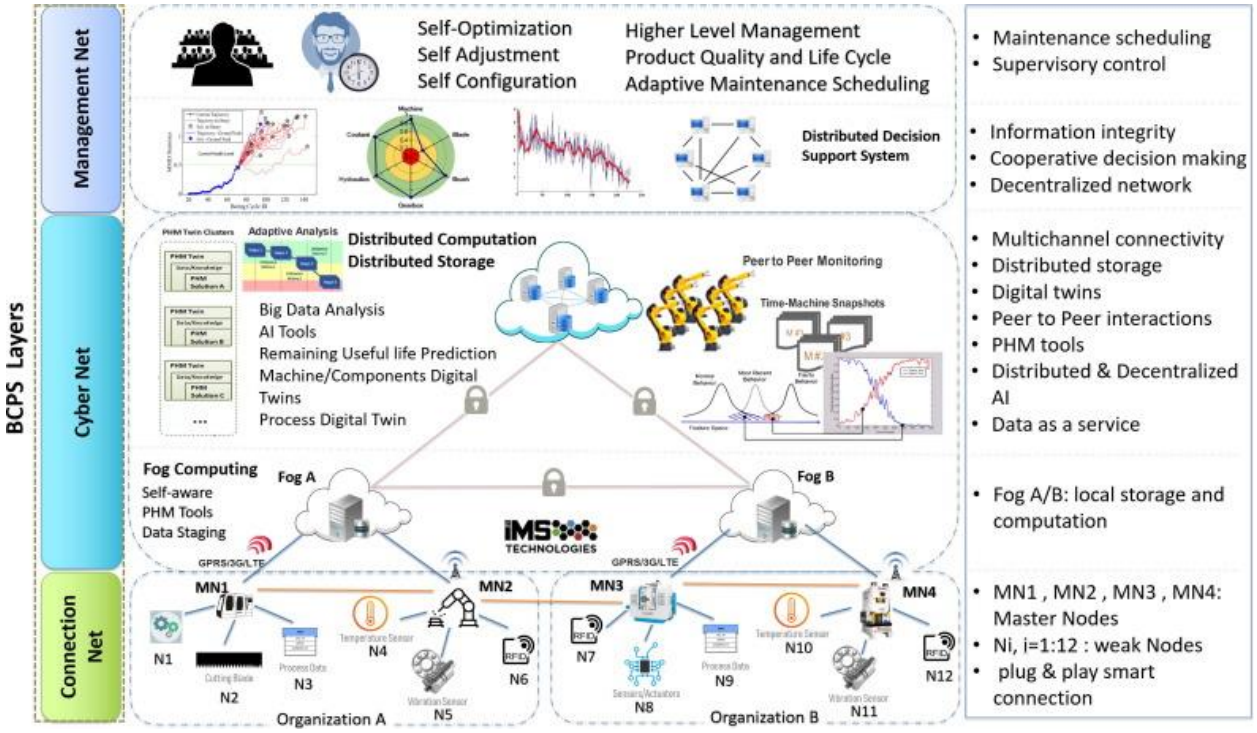
An Example: Blockchain-based Cyber Physical Systems (BPCS)

A unified three-level blockchain architecture is proposed as a guideline for researchers and industries to clearly identify the potentials of blockchain and adapt,

develop, and incorporate this technology with their manufacturing developments towards Industry 4.0.



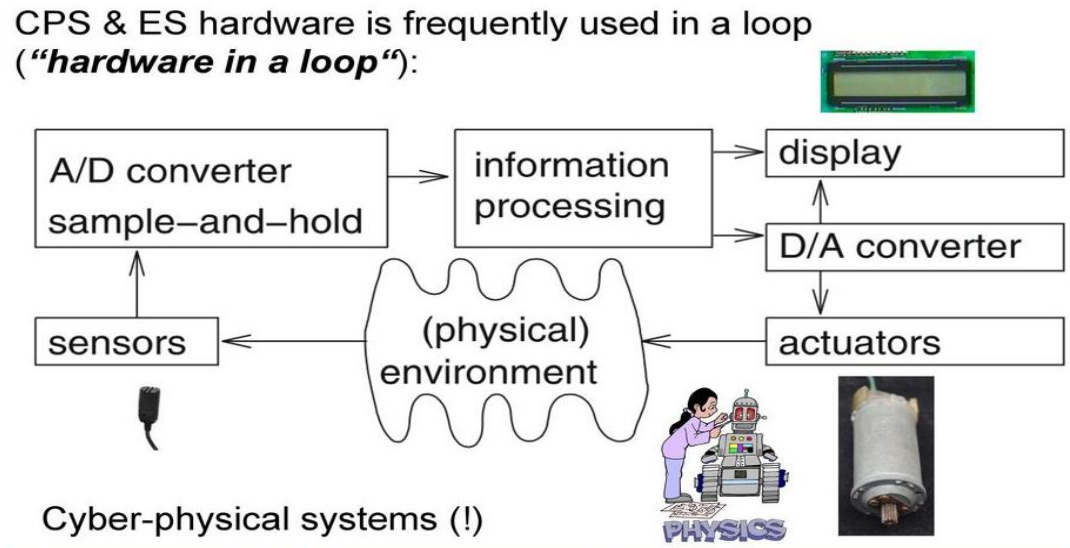
Decentralized computer architecture for CPPs



Distributed Computation and Storage in CPS

2.10 CYBER PHYSICAL SYSTEM HARDWARE PLATFORM

The European project, a three-year program which began in February 2015 and received 2.88 million euros (about $3.5 million) in funding from the European Commission under the Horizon 2020 program, addresses the lack of simulation tools and models for full system design and analysis. This is mainly because most existing simulation tools for complex CPS only efficiently handle parts of a system while mainly focusing on performance.



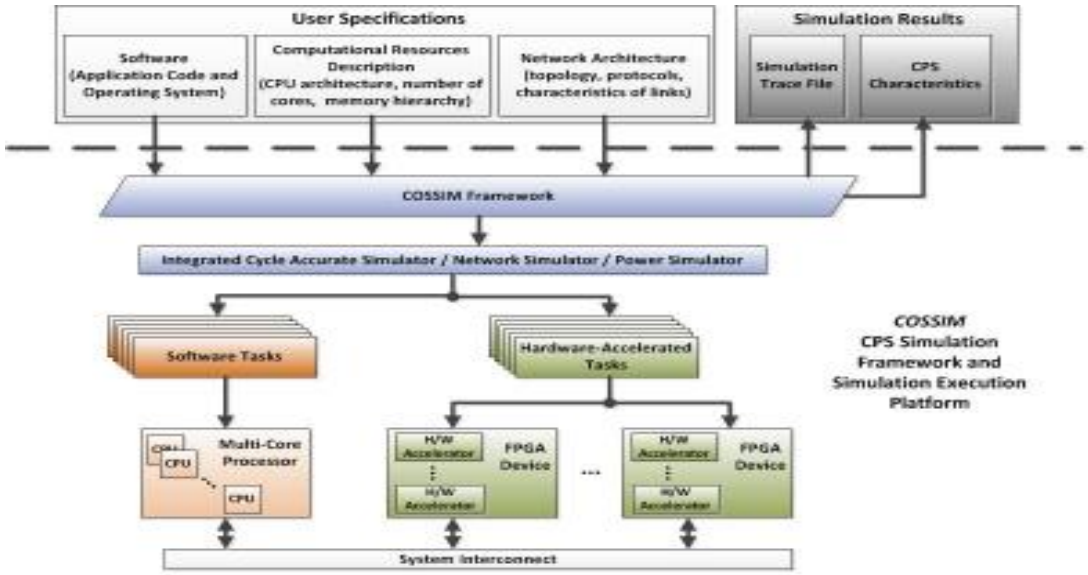
CPS Hardware

They also require extreme amounts of processing resources and computation time to accurately simulate the CPS nodes' processing. Faster approaches are available, however as they function at high levels of abstraction, they cannot provide the accuracy required to model the exact behavior of the system under design to guarantee that it meets the requirements in terms of performance and/or energy consumption.

COSSIM framework:

The new open source COSSIM framework, as it is called, developed in a R&D cooperation between STMicroelectronics with Politecnico di Milano, Telecommunication Systems Institute

and Synelixis, was successfully evaluated on an advanced client-server visual search use case that detects instances of objects. The evaluation included co-simulation with an experimental embedded system made by ST Microelectronics.



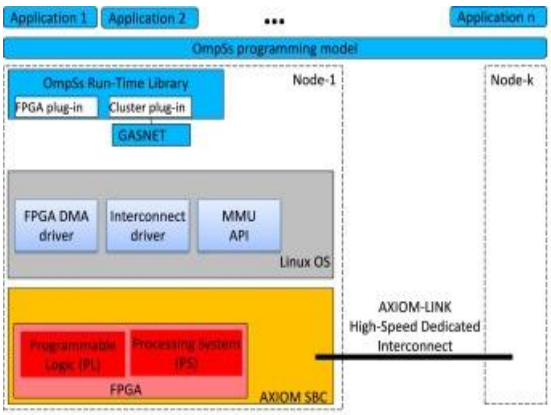
COSSIM framework

AXIOM:

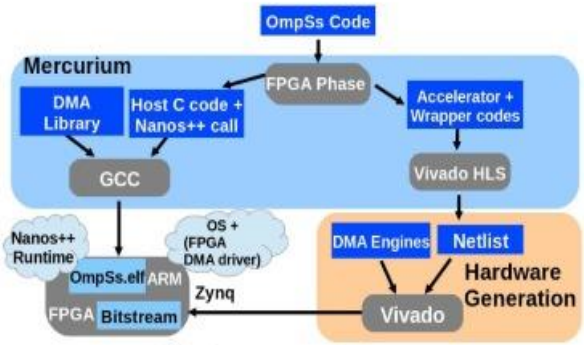
The AXIOM project (Agile, eXtensible, fast I/O Module) aims at developing a hardware-software platform for CPS such that i) it can use an easy parallel programming model and ii) it can easily scale-up the performance by adding multiple boards (e.g., 1 to 10 boards can run in parallel). AXIOM supports task-based programming model based on OmpSs and leverage a high-speed, inexpensive communication interface called AXIOM-Link. Another key aspect is that the board provides programmable logic (FPGA) to accelerate portions of an application. We are using smart video surveillance, and smart home living applications to drive our design.

The OmpSs programming model supports the execution of heterogeneous tasks written in OpenCL, CUDA, or a high-level C or C++ language that can be converted to the machine language used in GPUs or converted to the bitstream to program FPGAs. Also,

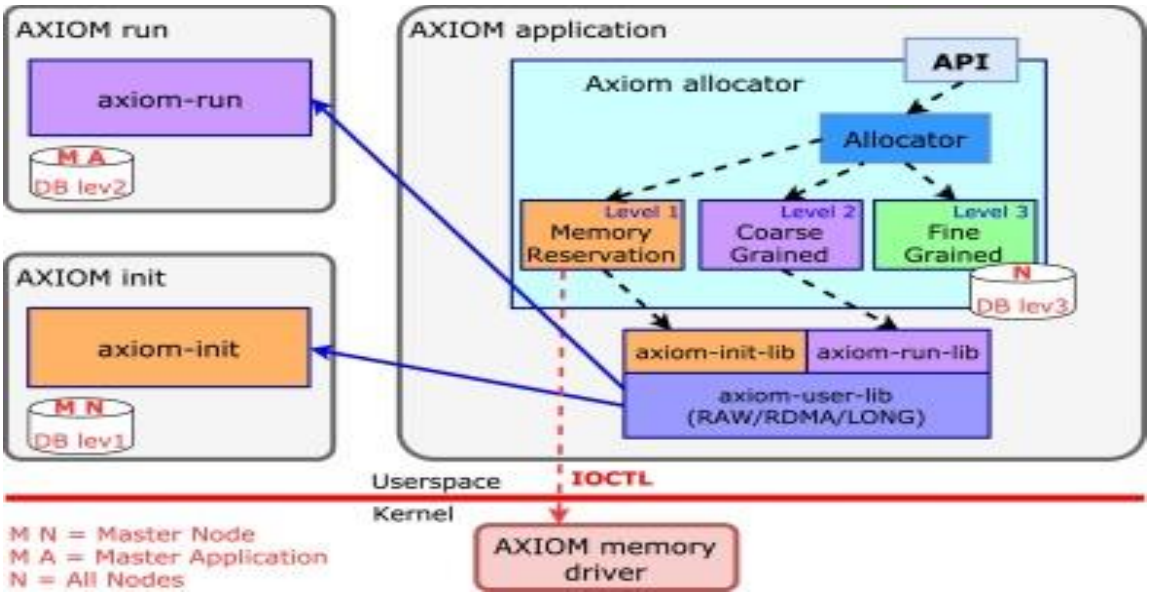
the runtime supports the communications within a cluster of distributed memory machines. OmpSs can target tasks to the different nodes of the cluster.



(a) AXIOM software stack



(b) FPGA programming support

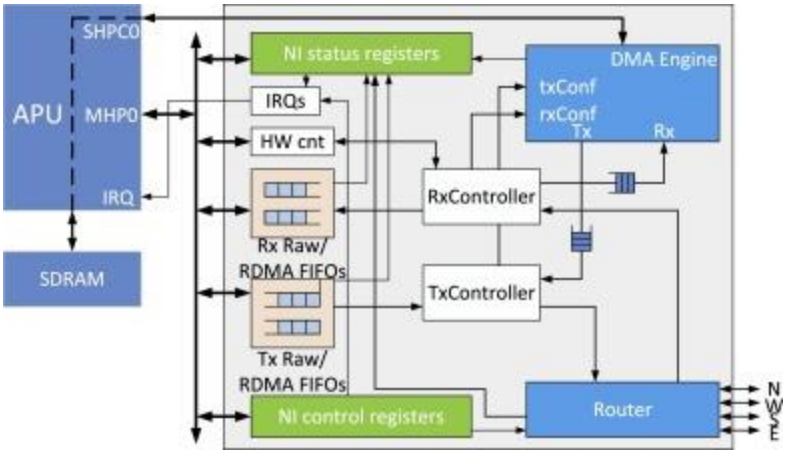


AXIOM platform for next-generation cyber physical systems

The AXIOM platform is designed around the Xilinx Zynq SoC (System on Chip) that features a multi-core ARM processor tightly coupled with FPGA fabric. AXIOM is designed to be modular at the next level, allowing the formation of more efficient

processing systems through low cost, but scalable high-speed interconnect. The latter will utilize the integrated gigabit-rate transceivers with relatively low-cost USB-C connectors to interconnect multiple boards.

Such connectivity will allow to build (or upgrade at a later moment) flexible and low-cost systems by cascading more AXIOM boards, without the need of costly connectors and cables. AXIOM boards will feature two or four bi-directional links, so that the nodes can be connected in many different ways, such as ring and 2D-mesh/torus.



AXIOM Network Interface Architecture

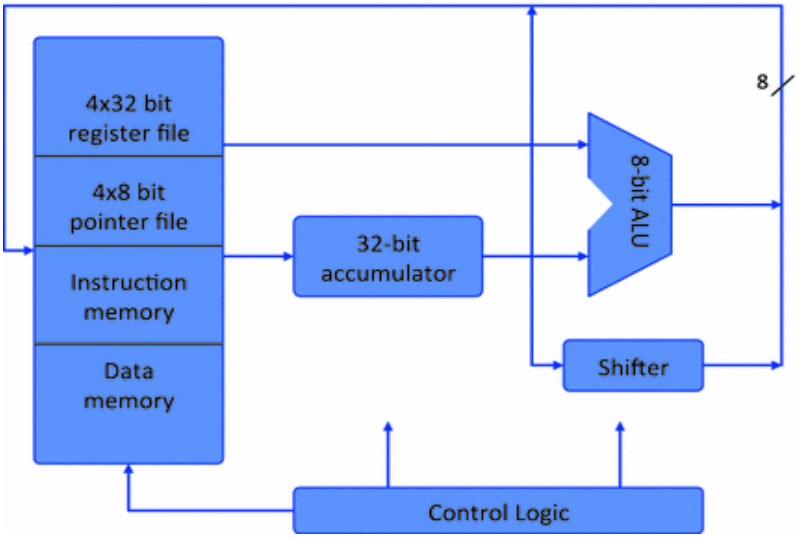
The AXIOM network interface architecture has remote direct memory access (RDMA) and remote write operations as basic communication primitives visible at the application level.

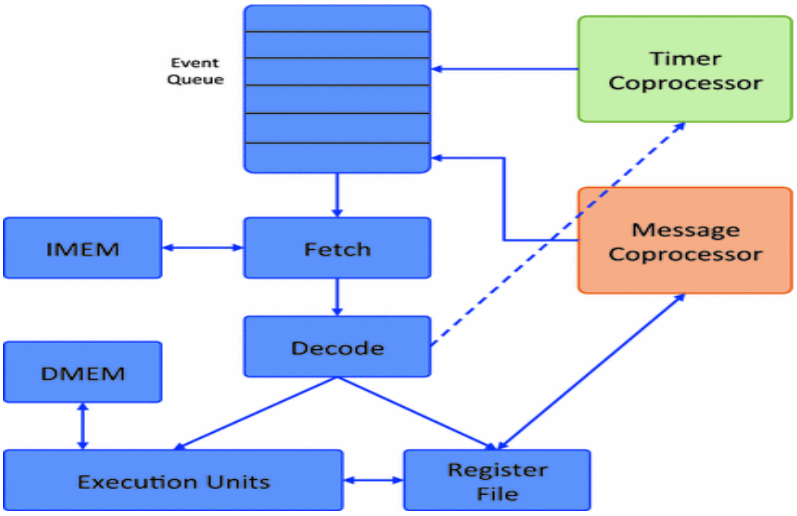
2.11 PROCESSORS, SENSORS, ACTUATORS - NETWORK

Processors

Communication processors are processors with specific optimizations to support communication systems. Communication processors exist in a wide variety of forms and can be categorized based on the communication system, such as wired or wireless and based on the layer in the communication system, which can be the physical layer, the medium, access, control layer, or the network layer. Communication processors can be categorized even more based on the application, such as audio, video or data and the end

system requiring the communication system such as a laptop, a cell phone, or a personal computer.





Network Processors in Wireless operation

SNAP Processor and Subliminal Processor are examples of processors used in wireless environment for functioning the CPS components.

Processor	Processor Speed	Bits	RAM	Power Consumption (mW)
Processor	Processor Speed	Bits	RAM	Power Consumption (mW)
TIMSP430 F2419	8 MHz	16	12 KB	8
Freescale MPC8313	50 MHz	8/32	128 Kb	8
ARM-OKI ML674K	33 MHz	16/32	512 Kb	145
Freescale-MPC8313	333 MHz	32	GB External	520
IMote 2.0	400 MHz	32	32 MB	574
Intel PXA255	400 MHz	32	64 MB	620
ADVANTICYS	1.8 GHz	32	2 GB	1

List of Processors used in Wireless Networks

Sensors

A sensor monitors environmental conditions such as fluid levels, temperatures, vibrations, or voltage. When these environmental conditions change, they send an electrical signal to the sensor, which can then send the data or an alert back to a centralized computer system or adjust the functioning of a particular piece of equipment. For example, if a motor reaches the temperature point of overheating, it can automatically shut off.

Types of sensors

- Temperature sensors
- Vibration sensors
- Security sensors
- Pressure sensors
- Humidity sensors
- Gas sensors

Actuators

An actuator, on the other hand, causes movement. It takes an electrical signal and combines it with an energy source to create physical motion. An actuator may be pneumatic, hydraulic, electric, thermal, or magnetic. For example, an electrical pulse may drive the functioning of a motor within an asset.

Types of Actuators:

- Manual Actuators

  These actuators require an employee to control gears, levers or wheels. Although they are inexpensive and simple to use, they have limited applicability.

- Pneumatic actuators

These actuators use gas pressure to power valves. The pressure pushes a piston to affect the valve stem.

- Hydraulic actuators

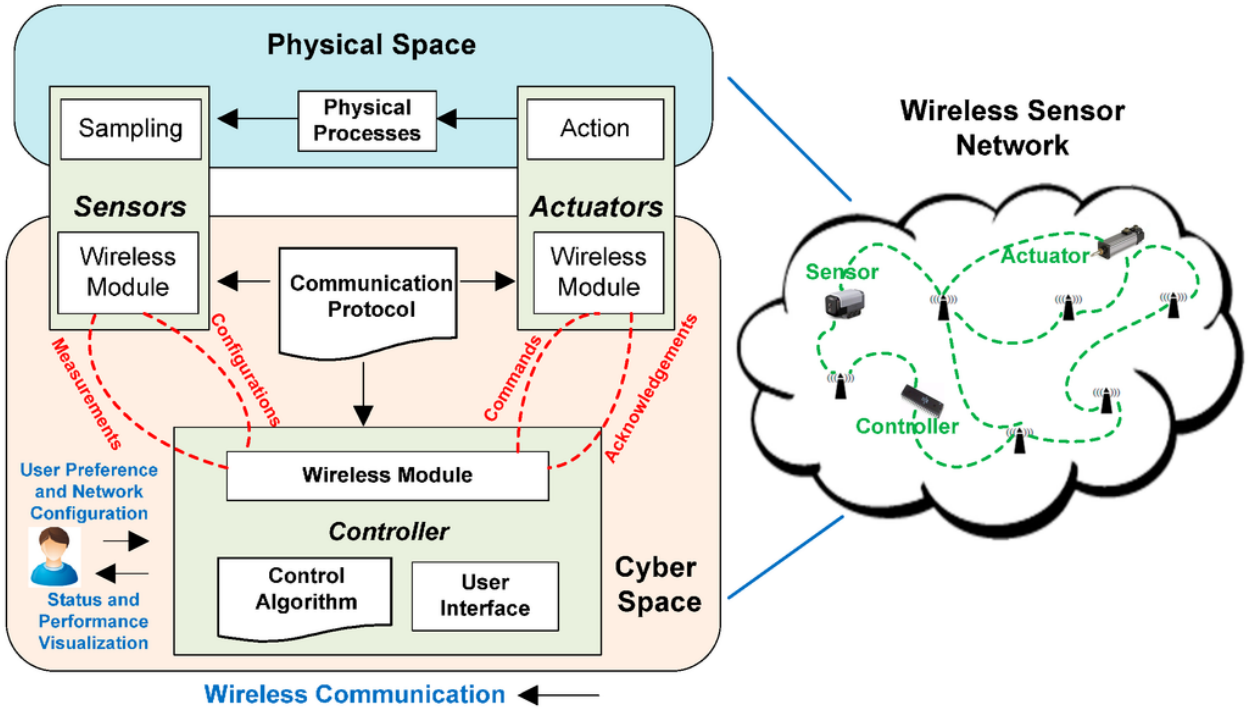
These actuators use fluid to generate pressure. Instead of using gas pressure, hydraulic actuators use fluid pressure to operate valves.

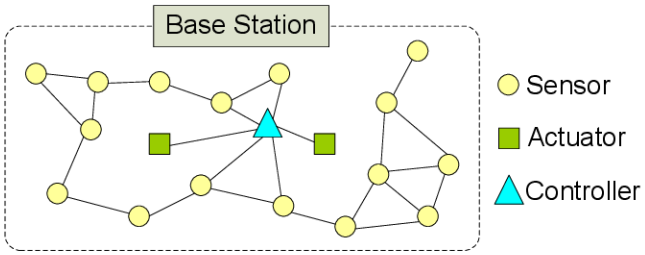
- Electric actuators

Electric actuators employ an electric motor to operate a valve. Although these actuators are quiet and efficient, they require batteries or electricity, which may not always be available in particular locations.

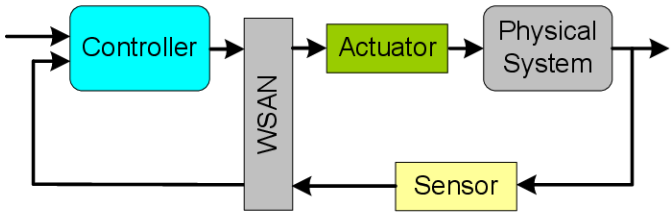
- Spring actuators

These actuators hold spring back until a trigger occurs. Once a particular threshold is reached, the spring releases and operates the valve. These are typically used in one-time emergency applications.





(a) Network topology



(b) Abstraction of control application

WSNs have gained importance in recent years due to the proliferation of Micro-Electro-Mechanical Systems (MEMS) technology facilitating the fabrication of Smart Sensors. The miniature sized sensors with limited computing power can sense, gather information and measure several environment parameters and, based on the local decision, forward it to a base station.

Though battery is the main source of power for these devices, recent techniques have started to focus on energy harvesting from natural resources like wind, solar energy and natural vibrations.

A typical WSN has little or no infrastructure and consists of numerous sensor nodes laid out over the monitoring area that gather data over time. For infrastructure based WSN, network entities like gateways, access points and network manager are required. Hence, a strict categorization for WSN deployment strategy is the structured and non-structured WSN. Depending on the environment, WSNs can be classified into terrestrial WSN, underwater WSN, underground WSN, multi-media WSN and mobile WSN. An unstructured WSN would consist of a dense collection of sensor nodes deployed in an *ad hoc* unattended manner.

. Sensors and actuators are necessary to interact with the physical world for data exchange, which is the most important feature of CPS, as they are responsible for sensing the conditions from the physical machines and environment, and executing control commands. Multiple sensors distributed on physical devices and in the environment, large-scale distributed data acquisition (e.g., material properties, real-time performance). Hierarchical levels of CPS and DTs in manufacturing. Through data management, processing, and analysis in the cyber world, control commands are generated based on predefined rules and the control semantic specification. The results are fed back to the actuators, which execute operations according to the control commands in order to adapt to changes. The data and control bus provides support for real-time communication and data exchange.

With sensors and actuators, any changes in physical process (e.g., behavior, conditions, or performance) cause changes in the cyber world, and vice versa. Therefore, the sensors and actuators can be considered to be the core elements of CPS. With sensors and actuators, any changes in physical process (e.g., behavior, conditions, or performance) cause changes in the cyber world, and vice versa. Therefore, the sensors and actuators can be considered to be the core elements of CPS.

Networked Cyber-physical system

Cyber-physical systems (CPS) are engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components. The communications and networking challenges associated with realizing CPS is often termed the

research area of Networked CPS. An increasing number and types of devices can sense and impact the environment around them. These devices can vary from small wireless communicating devices to large systems, communicating locally among themselves or over networks to cloud servers. The environments in which these devices will be expected to operate are diverse and range from operating underwater to flying in the air, being used in manufacturing facilities to being used for personal communications.

a) Cognitive Radios

b) Drone Systems

c) Underwater Robot Systems

d) Bluetooth Low Energy Devices

e) Wearable Devices

f) Advanced Manufacturing Systems and the Cloud

Participants in CPS

Realizing Networked CPS thus presents a rich set of research challenges that need to be addressed in areas such as indoor and outdoor positioning and localization, wireless communications and networking, security and privacy, cloud computing, mobile and wearable computing, and data analytics. The communications and networking challenges associated with Networked CPS go beyond those of just networked systems through a greater emphasis on the integration of physical components into systems.

2.12 Wireless Hart

As the need for additional process measurements increases, users seek a simple, reliable, secure and cost-effective method to deliver new measurement values to control systems

without the need to run more wires. With process improvements, plant expansions, regulatory requirements and safety levels demands for additional measurements, users are looking to wireless technology for that solution.

Wireless HART within telecommunications and computing, is a wireless sensor networking technology. It is based on the Highway Addressable Remote Transducer Protocol. Developed as a multi-vendor, interoperable wireless standard, Wireless HART was defined for the requirements of process field device networks.

Technical description

The protocol utilizes a time synchronized, self-organizing, and self-healing mesh architecture. The protocol supports operation in the 2.4 GHz ISM band using IEEE 802.15.4 standard radios. The underlying wireless technology is based on the work of Dust Networks' TSMP technology.

With approximately 30 million HART devices installed and in service worldwide, HART technology is the most widely used field communication protocol for intelligent process instrumentation. With the additional capability of wireless communication, the legacy of benefits this powerful technology provides continues to deliver the operational insight users need to remain competitive.

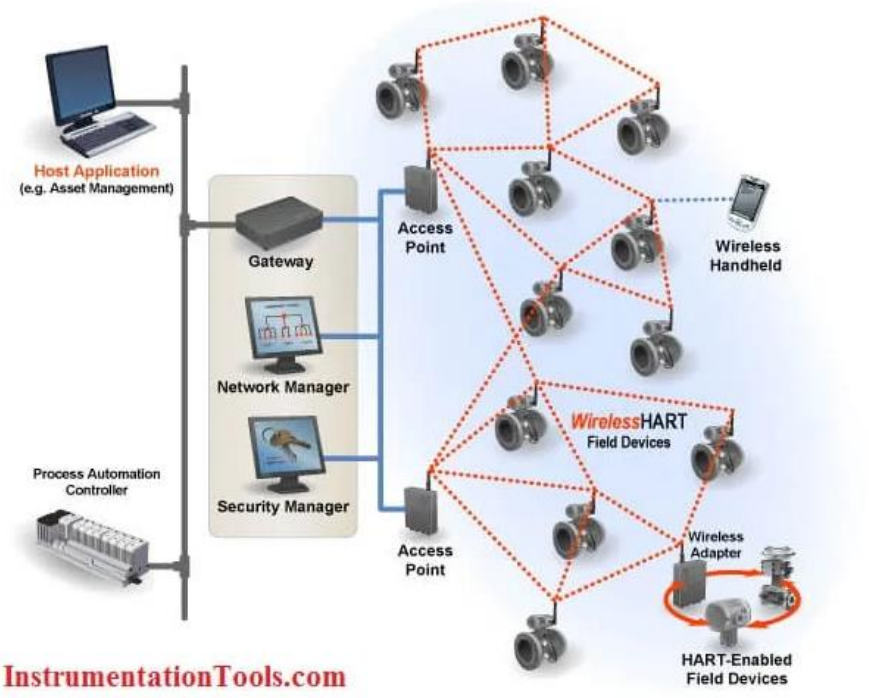
WirelessHART – How it works

*Wireless*HART is a wireless mesh network communications protocol for process automation applications. It adds wireless capabilities to the HART Protocol while maintaining compatibility with existing HART devices, commands, and tools.

Each *Wireless*HART network includes three main elements:

- **Wireless field devices** connected to process or plant equipment. This device could be a device with *Wireless*HART built in or an existing installed HART-enabled device with a *Wireless*HART adapter attached to it.

- **Gateways** enable communication between these devices and host applications connected to a high-speed backbone or other existing plant communications network.
- **A Network Manager** is responsible for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health. The Network Manager can be integrated into the gateway, host application, or process automation controller.



The network uses IEEE 802.15.4 compatible radios operating in the 2.4GHz Industrial, Scientific, and Medical radio band. The radios employ direct-sequence spread spectrum technology and channel hopping for communication security and reliability, as well as TDMA synchronized, latency-controlled communications between devices on the network. This technology has been proven in field trials and real plant installations across a broad range of process control industries.

Each device in the mesh network can serve as a router for messages from other devices. In other words, a device doesn't have to communicate directly to a gateway, but just forward its message to the next closest device. This extends the range of the network and provides redundant communication routes to increase reliability.

The **Network Manager** determines the redundant routes based on latency, efficiency and reliability. To ensure the redundant routes remain open and unobstructed, messages continuously alternate between the redundant paths. Consequently, like the Internet, if a message is unable to reach its destination by one path, it is automatically re-routed to follow a known-good, redundant path with no loss of data.

The mesh design also makes adding or moving devices easy. As long as a device is within range of others in the network, it can communicate.

For flexibility to meet different application requirements, the WirelessHART standard supports multiple messaging modes including one-way publishing of process and control values, spontaneous notification by exception, ad-hoc request/response, and auto-segmented block transfers of large data sets. These capabilities allow communications to be tailored to application requirements thereby reducing power usage and overhead.

Components of WirelessHART technology

A **Gateway** provides the connection to the host network. *Wireless*HART and then the main host interfaces such as Modbus – Profibus – Ethernet. The Gateway also provides the network manager and security manager (these functions can also exist at the host level – however initially they will be in the gateway)

The Network manager builds and maintains the MESH network. It identifies the best paths and manages distribution of slot time access (*Wireless*HART divides each second into 10msec slots) Slot access depends upon the required process value refresh rate and other access (alarm reporting – configuration changes) The **Security manager** manages and distributes security encryption keys. It also holds the list of authorized devices to join the network.

The **Process** includes measuring devices – the HART-enabled instrumentation.

A **Repeater** is a device which routes *Wireless*HART messages but may have no process connection of its own. Its main use would be to extend the range of a*Wireless*HART network or help "go around" an existing or new obstacle (New process vessel). All instruments in a *Wireless*HART network have routing capability which simplifies planning and implementation of a wireless network.

The **Adapter** is a device which plugs into an existing HART-enabled instrument to pass the instrument data through a *Wireless*HART network to the host. The adapter could be located anywhere along the instrument 4-20mA cable; it could be battery powered or obtain its power from the 4-20Ma cable. Some adapters will be battery powered and use the same battery to power the instrument as well – in this case there will be no 4-20mA signal to the host – all process data will be reported via *Wireless*HART

A **Handheld Terminal** may come in two versions. In the first case, the handheld will be a standard HART FSK configuration unit (just add new device DDs or DOF files), just like the one used for everyday tasks such as routine maintenance and calibration checks. In the case of wireless support, the handheld is used to join a new instrument to an existing *Wireless*HART network.

In the second case the handheld has a *Wireless*HART connection to the gateway and then down to an instrument and could be used for reading PV or diagnostics.

2.13 CAN,

What does CAN mean in automotive terms?

CAN stands for **controller area network**. They are designed specially to meet the Automobile Industry needs. Before CAN was introduced, each electronic device is connected to other devices using many wires to enable communication.

CAN stands for controller area network. They are designed specially to meet the Automobile Industry needs. Before CAN was introduced, each electronic device is connected to other devices using many wires to enable communication. But when the functions in the automobile system increased, it was difficult to maintain because of the tedious wiring system. With the help of the CAN bus system, which allows ECUs to communicate with each other without much complexity by just connecting each ECU to the common serial bus. Hence when compared with the other protocols used in automotive systems i.e., CAN vs LIN, CAN is robust due to less complexity.

CAN Protocol can be defined as a set of rules for transmitting and receiving messages in a network of electronic devices connected through a serial bus. Each electronic device in a CAN network is called a node. Each node must have hardware and software embedded in them for data exchange. Every node of a CAN bus system has a host microcontroller unit, CAN controller and, CAN transceiver in it. CAN controller is a chip that can be embedded inside the host controller or added separately, which is needed to manage the data and sends data via transceiver over the serial bus and vice versa. CAN Transceiver chip is used to adapt signals to CAN bus levels.

CAN is a message-based protocol where every message is identified by a predefined unique ID. The transmitted data packet is received by all nodes in a CAN bus network, but depending on the ID, CAN node decides whether to accept it or not. CAN bus follows the arbitration process when multiple nodes try to send data at the same time.

CAN Bus Electrical Specification:

CAN signals processed by CAN transceiver are single-ended signals and differential signals (CANH and CANL). CAN High and CAN Low lines are at 2.5v in ideal condition. CAN defines logic “zero” as the dominant bit and logic “one” as the recessive bit. When the dominant bit is transmitted, CAN High goes to 3.5v, and CAN low goes to 1.5v i.e., the differential voltage of the dominant bit is 2v. When the recessive bit is transmitted CAN High and CAN Low lines are driven to 2.5v, indicating the differential voltage of the recessive bit is 0v.CAN bus terminal resistor of 120 ohms should be added at the physical end of CANH and CANL lines to avoid any signal reflections.

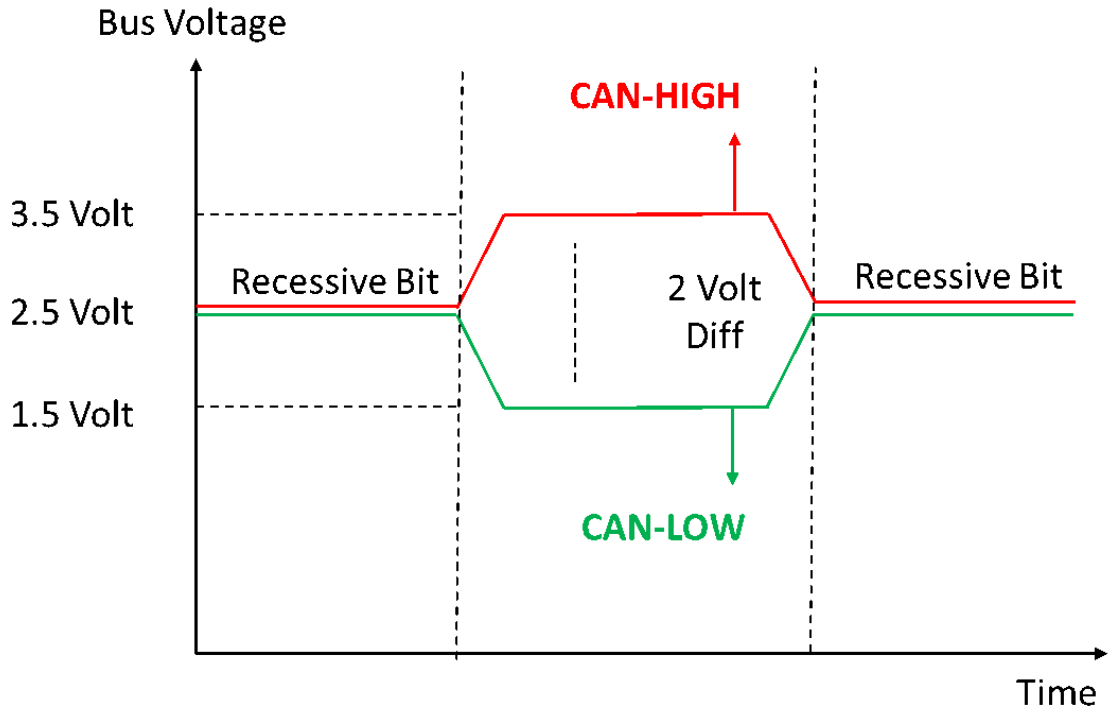


Figure 1. CAN bus differential signals

Layered architecture of CAN

It consists of three layers i.e. Application layer, Data link layer, and Physical layer.

- **Application Layer**: This layer interacts with the operating system or application of a CAN device.
- **Data Link Layer:** It connects actual data to the protocol in terms of sending, receiving, and validating data.
- **Physical Layer:** It represents the actual Hardware i.e. CAN Controller and Transceiver.

CAN physical layer Characteristics

The CAN physical layer is divided into three parts: Physical coding implemented in CAN controller chips, physical media attachment specifying Transceiver characteristics, physical media dependent sublayer which is application-specific and not standardized.

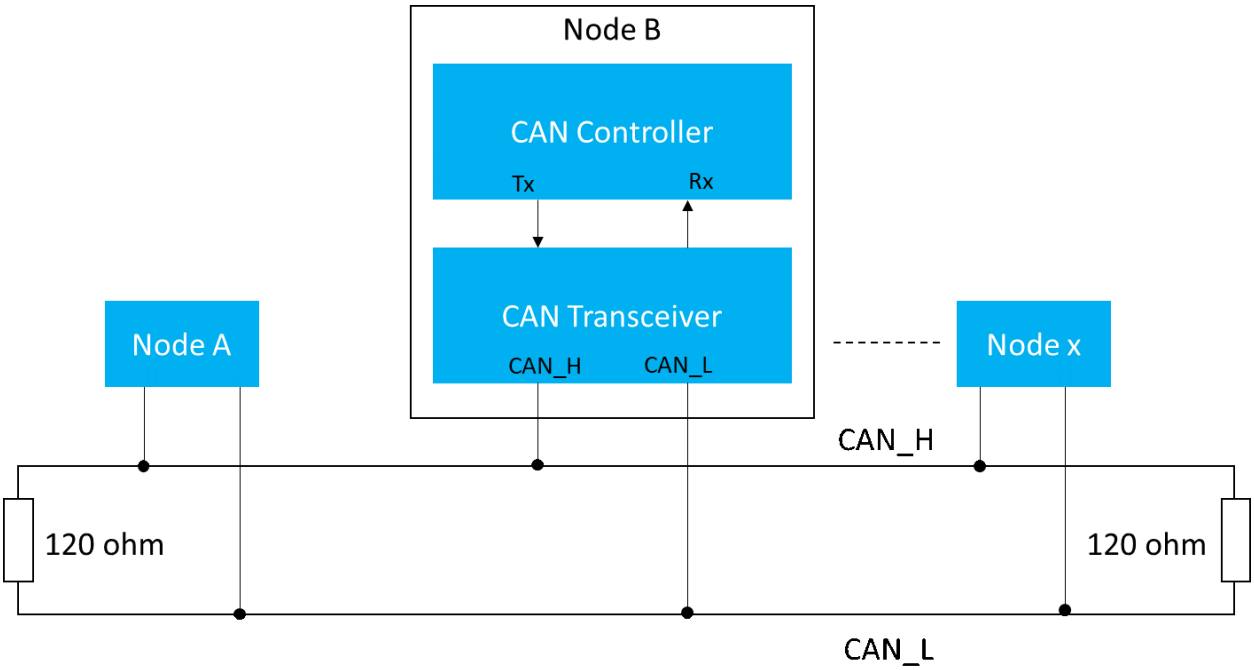


Figure 5. CAN bus Wiring Diagram

CAN bus support in various Microcontrollers:

CAN Bus shield for Arduino

- CAN bus Shield adopts CAN bus controller with SPI interface and CAN transceiver and provides CAN bus capabilities to Arduino.
- Arduino with CAN bus helps to get the information like vehicle speed, fuel consumption, Temperature from the ECU's
- An Arduino CAN library is used to send and receive CAN messages over CAN bus.

Raspberry Pi CAN bus:

Raspberry Pi does not have specific hardware i.e. CAN controller and CAN transceiver to support CAN protocol. CAN bus was not supported by Raspberry Pi Software. Raspberry Pi supports CAN communication through the SPI interface.

Raspberry Pi is connected to an external CAN Controller supported by the board through the SPI interface and CAN controller is connected to CAN transceiver through Rx and Tx lines.

Examples of CAN controller: SJA100, MCP2515
Examples of CAN Transceiver: TJA1040, MCP2551

STM32 CAN bus:

Unlike Arduino and Raspberry Pi, STM32 has a CAN controller embedded in it. STM provides CAN bus drivers and those API's can be used when GPIO configured either to CAN1, CAN2.

2.14 Automotive Ethernet –

The single twisted pair is a fundamental difference between standard Ethernet and automotive Ethernet. Unlike standard ethernet, with a dedicated transmit-and-receive path, automotive Ethernet, has a single twisted pair being used for both transmit and receive operations at the same time.

Automation and connectivity are driving the automotive industry forward, and in answer to the growing need for bandwidth, flexibility, and cost-effectiveness, in-car networks are evolving. Ethernet is emerging as a preferred choice.

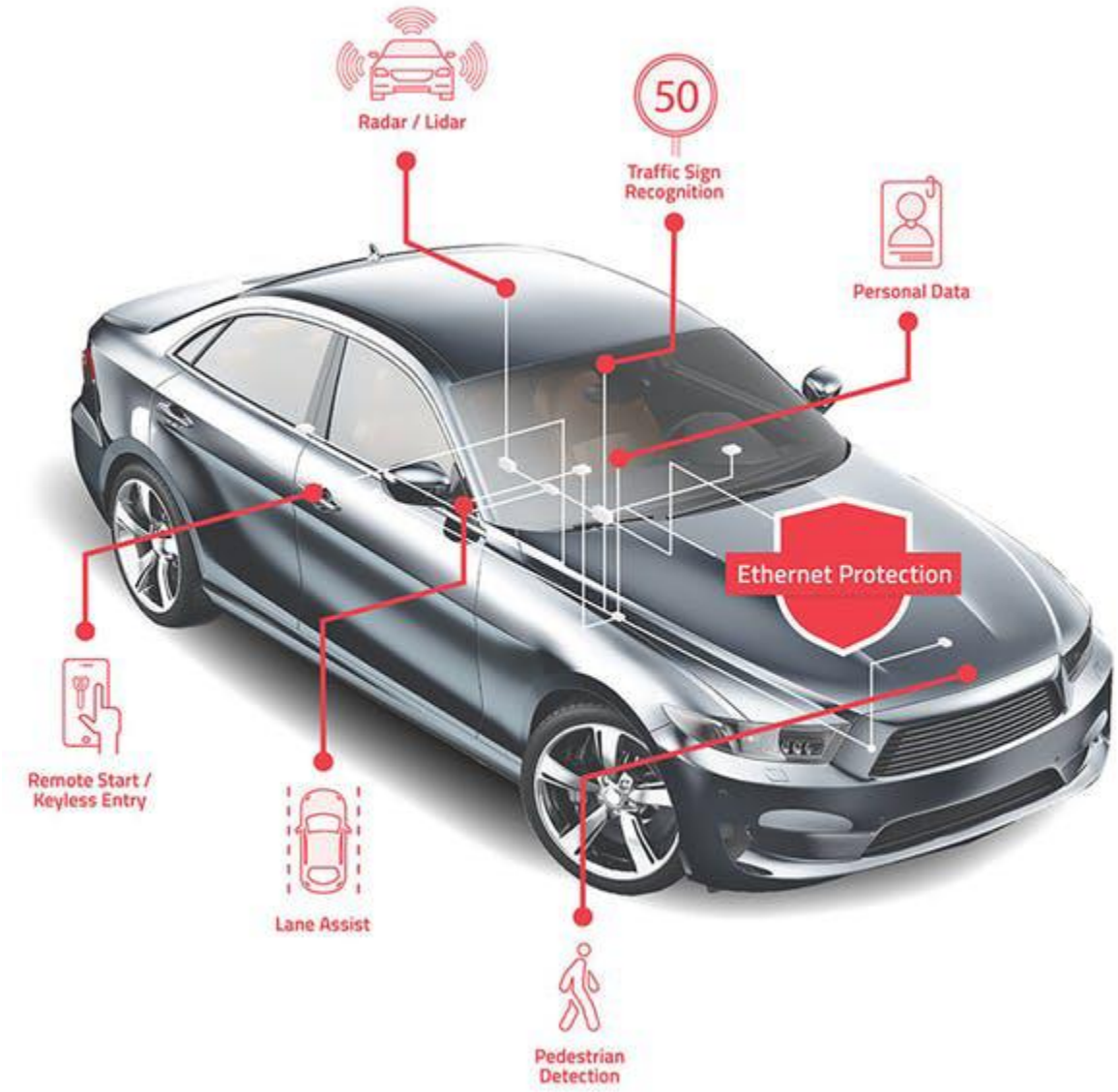
What is Automotive Ethernet?

Although Ethernet hasn't been widely used in the automotive industry until recent years, it's a mature technology with over 30 years of use in the wider networking market. Developed in the 1970s, it has become the standard for general computer networking around the world. A host of networking protocols and security methodologies have been developed in that time, lending themselves well to the challenges of automotive networking and cybersecurity.

Ethernet should be familiar to most of us. You might find yourself using it to connect your home computer to your router or modem, and if not, you will certainly be aware of Ethernet's cable-free counterpart, Wi-Fi. Automotive Ethernet is slightly different; a flavor of regular Ethernet, it's optimized for vehicular use.

Until now, it's been used primarily for diagnostics, in-vehicle-infotainment (IVI) systems, and connecting remote sensors. Data-heavy, these systems require greater bandwidth to transmit data

at the speeds necessary to maintain driver safety—speeds that networks such as CAN and FlexRay are unable to provide. When you consider the growing interest in autonomous vehicles and the connectivity they will require, you begin to see the benefits automotive Ethernet can offer.



What are the benefits?

Autonomous vehicles (AVs) will require a host of connectivity features to function effectively, such as cameras, LiDAR, and traffic-sign recognition. These sensors, which enable vehicle-to-

everything (V2X) connectivity, are vital to their success. Thus, the demand for greater bandwidth is set to skyrocket.

Designed to accommodate this demand and offering speeds of up to 100 Mb/s in its current form, Ethernet is soon to reach faster speeds. The IEEE802.3 working group, responsible for automotive Ethernet, is working on a much faster multi-gig standard for the future. Contrast this to the kilobit-per-second and low megabit-per-second speeds offered by CAN and LIN, and you see its appeal.

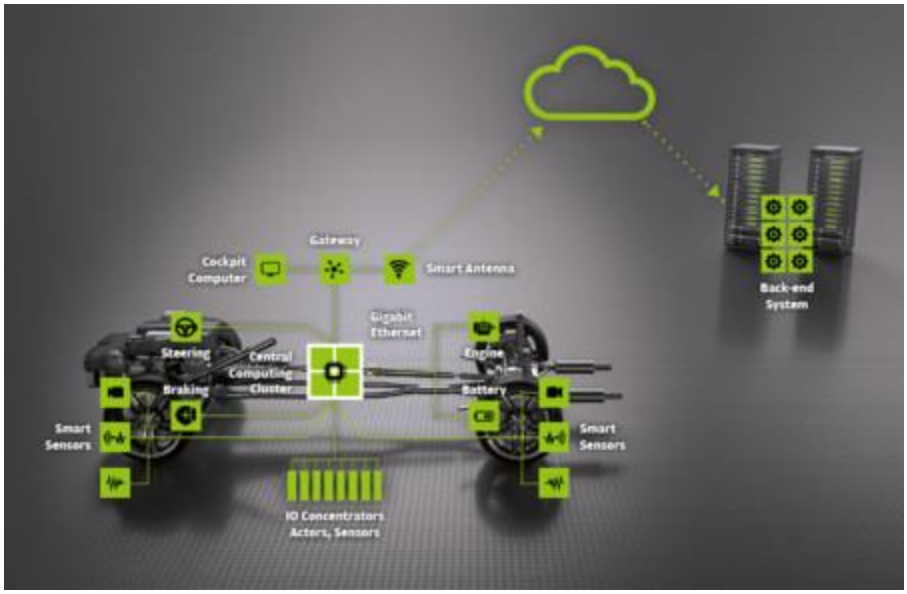
It has a rival in Media Oriented Systems Transport (MOST), a network that has been primarily used for infotainment and media systems. MOST offers 100- to 150-Mb/s speeds; however, proprietary licensing, restricted access to hardware, and reliance on heavy coax cables, or easily damaged optical fiber, have limited its market.

Ethernet offers greater future potential. It's built for bandwidth, is available from a wealth of potential providers, and with switch networking, offers greater scalability. It's also a lightweight and cost-effective solution, using single unshielded twisted-pair (UTP) cabling. Broadcom, the company responsible for introducing the current automotive standard, BroadR-Reach, estimate they can reduce connectivity costs by 80% and cable weight by 30%.

When you consider the cost benefits, alongside the obvious compatibility advantages when connecting vehicles to smart infrastructure, it makes sense that Ethernet would lead the charge in future V2X connectivity. However, increased compatibility with existing infrastructure and networking methods creates new challenges in cybersecurity.

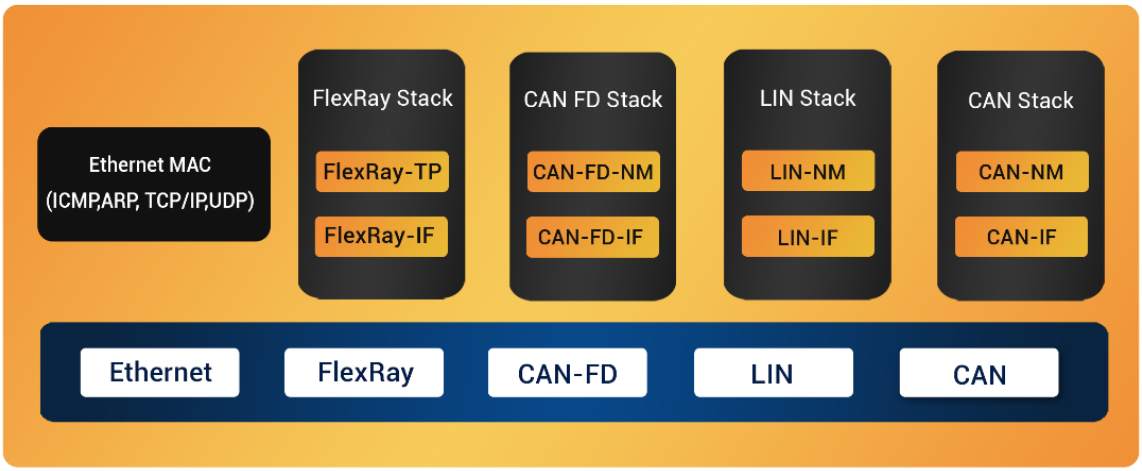
2.15 SOFTWARE STACK

The current Electric/Electronic (E/E) architecture in the vehicle integrates one or a few vehicle functions per control unit. This increases both the number of control units and distributed software functions and the complexity of connectivity respectively. The E/E architecture must perform an increasing number of driver assistance functions.



For less time-critical systems with more complex multi-core processors and different external interactors (software update, user input), dynamic operating systems have their strengths. The most important application scenarios are:

- Support of reconfiguration during run-time
- Service-oriented services and communication
- Partial software updates
- Simplification of software development by using POSIX interfaces instead of statically generated
- XML-based interface descriptions



ECU Communication Protocol Stacks

The protocol stacks can be readily integrated with any Automotive ECU and tooling applications. The utilized configuration **and end-of-line tools ensure faster configuration and integration of the protocol stacks.**

FlexRay (ISO 10681-1:2010)

- A deterministic and composable dual-channel ECU communication protocol
- Best suitable for applications that stipulate higher bandwidth e.g. Body Control Module, ADAS etc.
- We provide configurable FlexRay IF and Transport Layer and FlexRay drivers for industry-grade MCU platforms

CAN FD (ISO)

- Faster and flexible upgrade to Classical CAN
- Ideal communication protocol for modern solutions like Telematics, ADAS, Infotainment System

LIN Stack

- **LIN Bus Protocol stack** solutions are provided for LIN Master and LIN Slave.

- Deployed for roofing system, central locking system, wipers, etc.

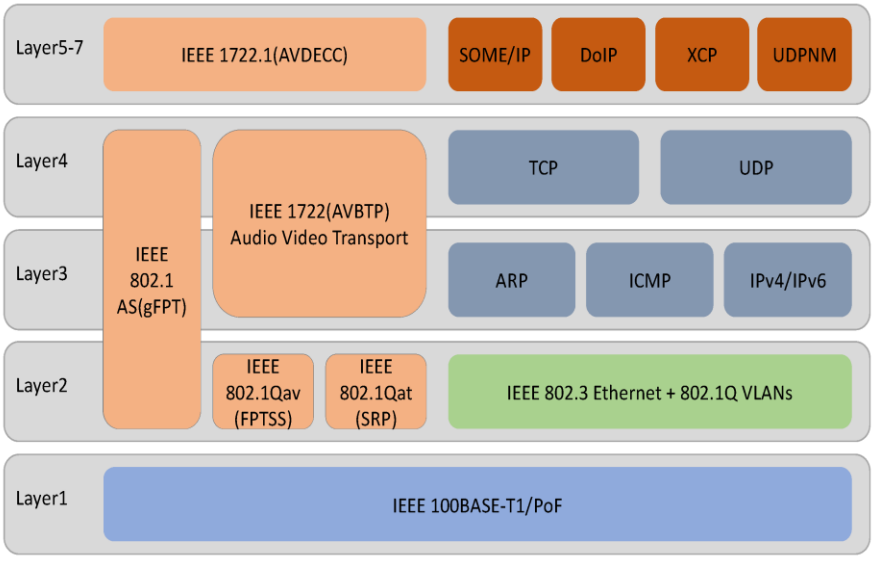
CAN 2.0

- Used as the choice communication protocol in passenger vehicles
- Our proprietary PC based CAN configuration tool reduces the time-to-market considerably.

Protocol Architecture

CAN bus is currently the most widely used in-vehicle bus protocol with the characteristics of low cost, high reliability, and real-time operation. However, with the development of the automotive electrical/electronic architecture and the increase in the amount of interactive data, the demand for network bandwidth in automotive applications has shown explosive growth.

Additionally, automotive Ethernet has become an important development direction for in-vehicle networks. The development of automotive Ethernet relies heavily on the standardization promotion work of some alliances, such as IEEE, OPEN, AUTOSAR, and AVnu.



Automotive Ethernet and its upper-layer protocol architecture

The current main automotive Ethernet and the upper-layer protocols supported by it are drawn in the ISO/OSI seven-layer architecture as shown in figure. There are three representative achievements in the physical layer of automotive Ethernet:

- BroadR-Reach technology of Broadcom Corporation, the AVB/TSN technology of AVnu Alliance, and TTEthernet of TTTech, among which BroadR-Reach technology has been standardized as 100BASE-T1 by IEEE802.3bw, which is also called OABR (OPEN Alliance BroadR-Reach).
- Above the physical layer, some standard protocols such as IEEE 802.1AS, IEEE 802.1Qat, IEEE 802.1Qav, IEEE 1722 (AVBTP) and IEEE 1722.1 (AVDECC) can be used to implement Ethernet AVB transmission.
- AVB enhances the real-time performance of traditional Ethernet audio and video transmission by adding precision clock synchronization and bandwidth reservation based on traditional Ethernet, which is a real-time audio and video transmission technology for vehicle systems.

Application layer protocols in Automotive Ethernet

Automotive Ethernet adopts the IEEE 802.3 interface standard, which can seamlessly support the widely used TCP/IP protocol cluster without adaptation. The corresponding application layer protocols include SOME/IP, Do/IP, XCP, UDPNM, etc. Compared with the traditional CAN bus, which is a signal-oriented communication method,

- SOME/IP is a service-oriented communication method.
- DolP is a diagnostic transmission protocol based on Ethernet that can encapsulate UDS and transmit it based on the IP network.
- XCP is mainly used for calibration, measurement, small-scale programming, and flashing.
- UDPNM is a network management protocol based on automobile Ethernet developed by
- AUTOSAR, which can effectively realize the coordinated sleep and wake-up of automobile Ethernet nodes.

SOME/IP is one of the core protocols in automotive Ethernet technology and uses SOA software development logic to achieve isolation and modular design. SOME/IP was first proposed by the BMW Group in 2011.

2.16 REAL-TIME OPERATING SYSTEM (RTOS)

RTOS is a multitasking operating system intended for real time applications. In the OS, there is a module called the scheduler, which schedules different tasks and determines when a process will execute on the processor. Multi-tasking is achieved in this way. The scheduler in RTOS is designed to provide a predictable execution pattern. In an embedded system, a certain event must be carried across strictly defined time.

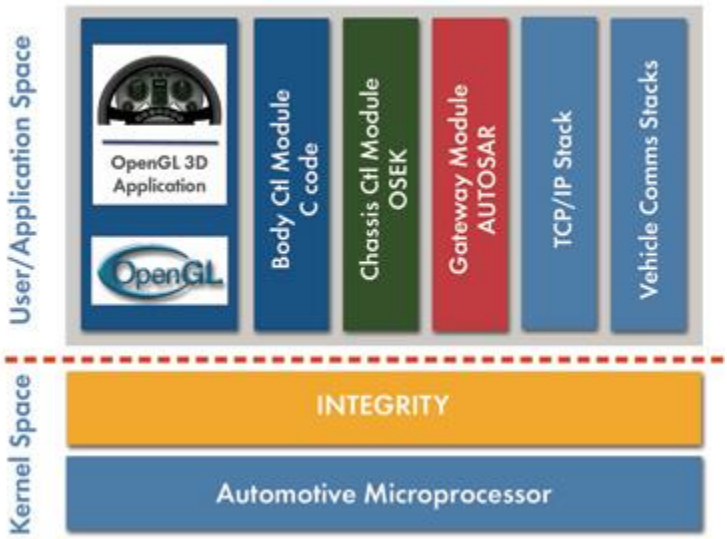
To meet real time requirements, the behavior of scheduler must be predictable. This type of OS which have a scheduler with predictable execution pattern is called real time OS (RTOS). The features of an RTOS are:

1. Context switching latency should be short.
2. Interrupt latency should be short.
3. Interrupt dispatch latency should be short.
4. Reliable and time bound inter process mechanism.
5. Support kernel pre-emption.

RTOS Responsibility

RTOS consists of the following main responsibilities:

1. Task management and scheduling
2. (Deferred) interrupt servicing
3. Inter-process communication and synchronization
4. Memory management



AUTOSAR support to INTEGRITY RTOS

RTOS Types:

- Hard real-time – Systems where it is absolutely imperative that responses occur within the required deadline. E.g. Flight control systems.

- Soft real-time – Systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. E.g. Data acquisition system.

- Real real-time – Systems which are hard real-time and which the response times are very short. E.g. Missile guidance system.

- Firm real-time – Systems which are soft real-time but in which there is no benefit from late delivery of service.

A single system may have all hard, soft and real real-time subsystems.

Automotive Operating System OSEK/VDX:

AUTomotive Open System ARchitecture (AUTOSAR) is **a development partnership of automotive interested parties founded in 2003**. It pursues the objective to create and establish an open and standardized software architecture for automotive electronic control units (ECUs). **AUTOSAR OS** specification uses the industry standard OSEK OS (ISO 17356-3) as the basis for the AUTOSAR OS.

OS Objects:

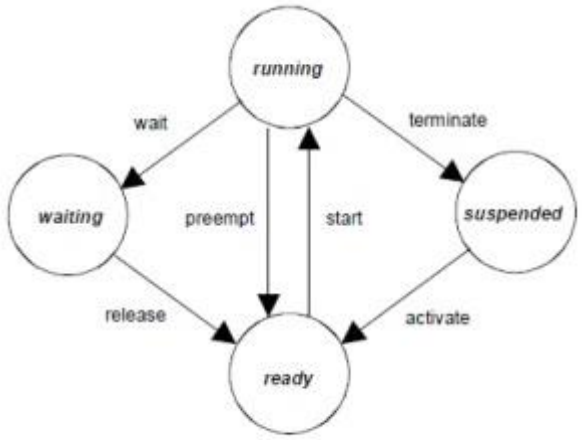
Task, Events, Counter, Scheduler, Resource, Alarm and Hook Functions are OS Objects.

Task:

A task provides the framework for the execution of functions. Complex control software can conveniently be subdivided in parts executed according to their real-time requirements. Two different task concepts are provided by the OSEK operating system:

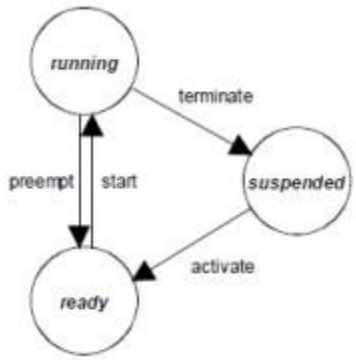
Extended task:

Extended tasks have four task states: 1. Running 2. Ready 3. Waiting 4. Suspended.



Basic Task:

Basic tasks have four task states: 1. Running 2. Ready 3. Suspended. Basic tasks do not have a *waiting* state.



SCHEDULING AND TYPES:

Four types of scheduling policy:

1. Pre-emptive
2. Non-Pre-emptive
3. Group of Task
4. Mixed Pre-emptive

RTOS uses pre-emptive scheduling. In pre-emptive scheduling, the higher priority task can interrupt a running process and the interrupted process will be resumed later. Certain functions in RTOS scheduling are:

EVENTS:

Assigned to Extended Task. Meaning of events is defined by the application, e.g. signalling of an expiring timer, the availability of a resource, the reception of a message, etc.

COUNTER:

Represent count value. Belong to Os-App. Increment by core on Os-App resides. Use to drive scheduler and alarm if they are in same core.

ALARM:

Use to activate task, Set event, increment counter and call-back alarm.

HOOK FUNCTIONS:

Hook routine are only used during debugging and are not used in a final product.

AUTO START OBJECTS:

Before scheduling starts, the Multi-Core OS shall activate all configured auto-start objects on the respective core.

OS Application:

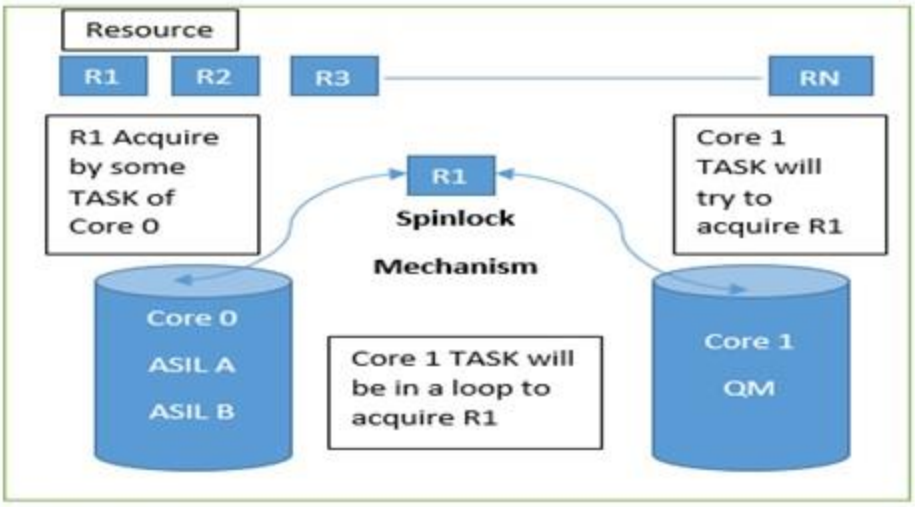
OS must be capable of supporting a collection of Operating System objects (Tasks, ISRs, Alarms, Schedule tables, Counters) that form a cohesive functional unit. This collection of objects is termed an OS-Application.an OS-Application may belong to different AUTOSAR Software Components. Type: Trusted and Non Trusted.

Trusted OS-Applications are allowed to run with monitoring or protection features disabled at runtime. Non-Trusted OS-Applications are not allowed to run with monitoring or protection features disabled at runtime. Trusted OS-Applications can be permitted access to IO space, allowed direct access to the hardware etc.

Spin lock:

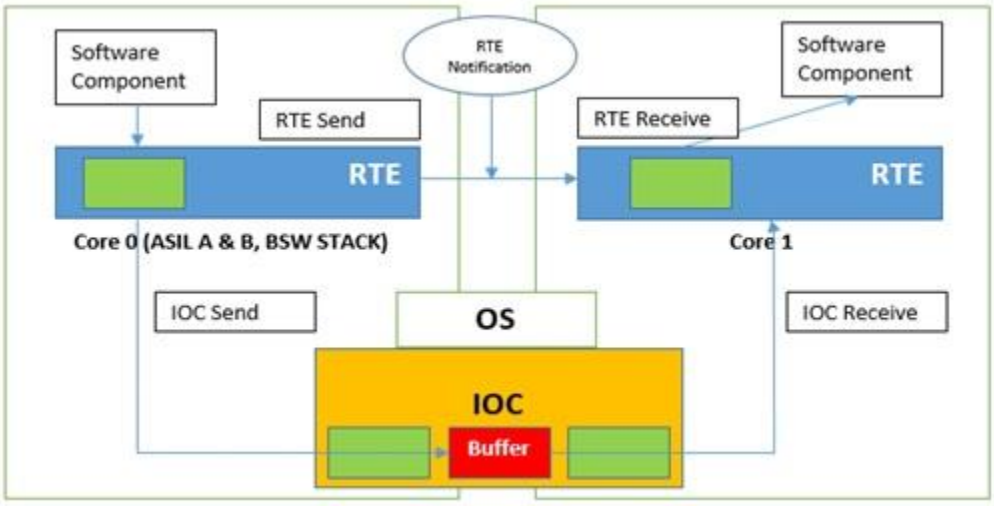
If a resource is locked, a thread that wants to access that resource may repetitively check whether the resource is available. During that time, the thread may loop and check the resource without doing any useful work. Suck a lock is termed as spin lock.

Used in multicore concept. Busy waiting mechanism that polls a (lock) variable until it becomes available. Once a lock variable is occupied by a TASK/ISR, other core shall be unable to occupy the lock variable. Spinlock will not reschedule these other tasks while they poll the lock variable.



IOC (Inter OS Communication):

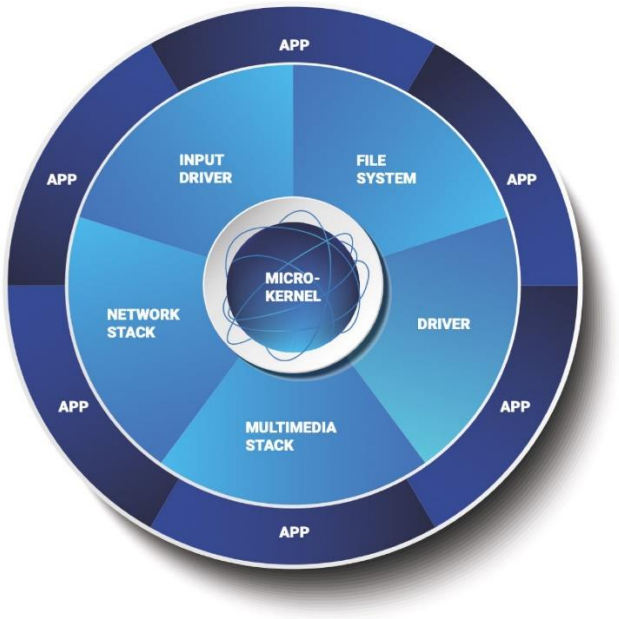
The “IOC” is responsible for the communication between OS-Applications and in particular for the communication crossing core or memory protection boundaries. Its internal functionality is closely connected to the Operating System.



Microkernel RTOS

A microkernel RTOS is structured with a tiny kernel that provides minimal services. The microkernel works with a team of optional cooperating processes that run outside the kernel space (in the user space), which provides higher-level OS functionality.

Microkernel OS Architecture



The microkernel design and modular architecture enable BlackBerry QNX customers to create highly optimized and reliable systems with low total cost of ownership. With the QNX Neutrino RTOS, embedded systems designers can create compelling, safe and secure devices

2.17 Scheduling Real Time control tasks

Timing issues are critical in real-time systems such as robot control, flight control, on-line multimedia systems, and real-time stock trading system, etc. Many real-time scheduling algorithms such as RM (Rate Monotonic), EDF (Earliest Deadline First), and LST (Least Slack time First) deal with CPU resources and network bandwidth scheduling to maximize real-time performance. As CPS (cyber-physical system) such as avionics, transportation, manufacturing processes, energy, healthcare, in which computers and physical systems are tightly coupled and timing is critical, is fast growing, real-time scheduling for CPS become new research issues in the real-time systems.

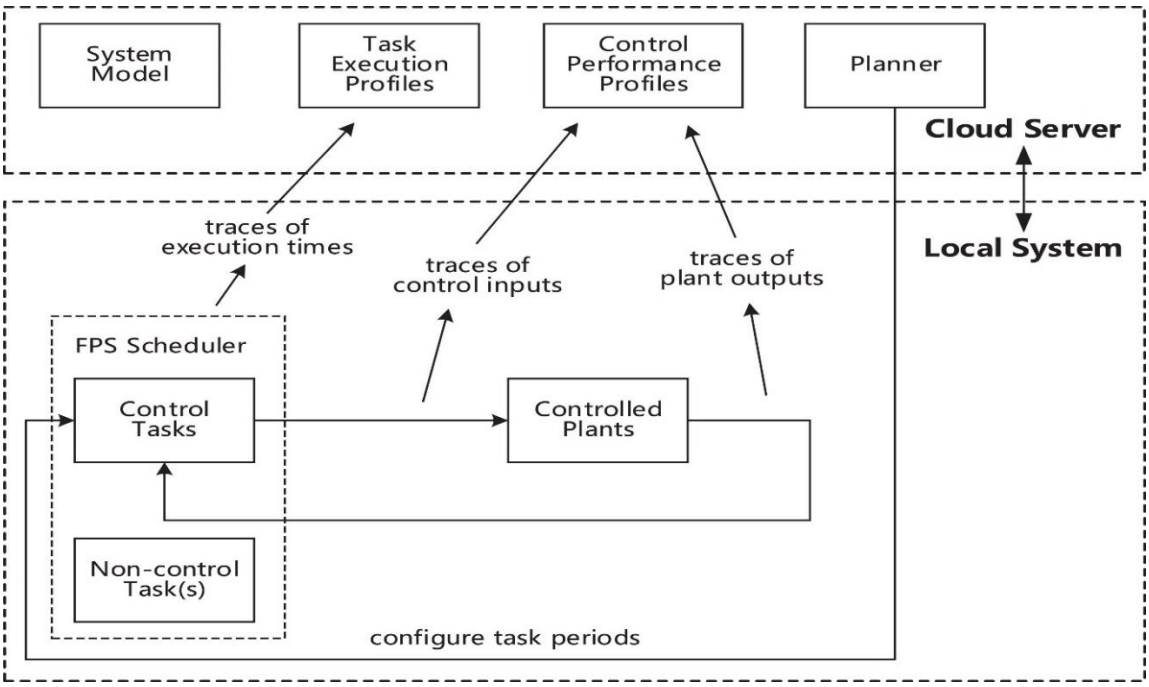
Many real-time scheduling algorithms have been proposed and widely used. However, in a cyber-physical system), we need to consider physical space (location, movement, etc.) as well cyber space (CPU, network, storage systems). Important real-time scheduling issues in CPS systems are as follows:

- ✓ **Spatial issues:** effective release time and deadline of real-time tasks may be different depending on location and physical movement delay of nodes participating in CPS. Realtime scheduling algorithms have to be modified to include spatial factors.
- ✓ **Conventional cyber real-time system** schedules CPU or network bandwidth. However, in real-time scheduling for CPS, location is matter. Location of nodes in CPS may affect on effective release time and deadline.

Difference between Conventional Real-Time scheduling and Real-Time scheduling for CPS

Functions	Conventional Real-Time scheduling	Real-Time scheduling for CPS
Scheduling resources	CPU, BW, memory, I/O	Servicing node
Scheduling environment	Cyber environment	Cyber and physical environment
Scheduling parameters	Cyber factors (Period, execution time, release time, deadline, etc.)	Cyber factors and physical factors (period, execution time, release time, deadline, migration delay time, etc.)
Migration	No migration is required for CPU and Job. *Consider CPU is serving node and Job is serviced node	Migration is required for CPU and Job
Well-known Scheduling algorithm	RM, EDF, LST	None
Spatial issues	Do not consider spatial issues	Physical migration delay time (between servicing node to serviced node)
Considering issues	Execution time, release time, deadline, laxity	Execution time, release time, effective deadline (deadline—moving time), effective laxity (laxity—moving delay time)

Real-Time Control Tasks with Fixed-Priority Scheduling (FPS)



General assumptions on the properties of the deployed FPS system are applied:

- The embedded computer (local system) consists of a uniprocessor and a preemptive scheduler using fixed-priority scheduling (FPS)

- All tasks in the system are released periodically and are initially schedulable, given the control tasks are using periods that can satisfy control specifications. The system has the ability to monitor task execution and response times, which is often supported by POSIX-compliant kernels and real-time programming languages such as Ada.

- The system itself has limited resources but has connectivity to a more powerful cloud server, e.g., IBM Bluemix, Amazon Web Services (AWS) or Google Cloud. The connection link does not need to be reliable or timing predictable.

Real-time scheduling Model

Assume parameters for real-time systems as follows:

- l_i: slack (laxity) time of task i (exponential distribution)
- e_i: execution time of task i (even distribution)
- m_i: migration time of servicing(computing) node to task (serviced node) i

Scheduling Types:

LST (Least Slack Time first):

- When task A and task B conflict, a task with least slack time is scheduled first.
- When we assume that the slack time of task A is shorter than that of task B, the slack time of task A is the exponential distribution ($2\lambda e^{-2\lambda t}$) of average $1/(2\lambda)$ while the slack of task B is the exponential distribution of $2\lambda e^{-\lambda t} - 2\lambda e^{-2\lambda t} = 2\lambda e^{-\lambda t}(1 - \lambda e^{-\lambda t})$.

ELST (Effective Least Slack Time first):

- Preemptive LST is an optimal algorithm in real-time scheduling algorithm. However, in CPS, we need to consider physical environments to improve the deadline meet ratio.

O-ELST (Optimal Effective Least Slack Time first):

- As the ELST algorithm cannot improve the real-time performance, an optimal algorithm which changes schedule for two tasks (task A and task B) when the changed schedule can improve the real-time performance is considered.

H-ELST (Heuristic ELST):

- Reduces time complexity while maintaining the deadline meet ratio.

Real-Time H-ELST scheduling algorithm is as follows:

- ❖ Modify the LST scheduling algorithm by weighting on the slack time and physical migration time;
- ❖ Give priority to serviced nodes with not only small slack time but also small moving time: as the weighted sum of slack time and moving time decreases, the priority increases;
- ❖ Give ‘‘α’’ ($0 < \alpha < 1$) weight to the slack time and ‘‘$1 - \alpha$’’ weight to the migration time. (Performance will depend on the value of the weight parameter α;)
- ❖ Focus on physical factors (migration delay time) as well as cyber factors (period, execution time, release time, deadline, etc.)