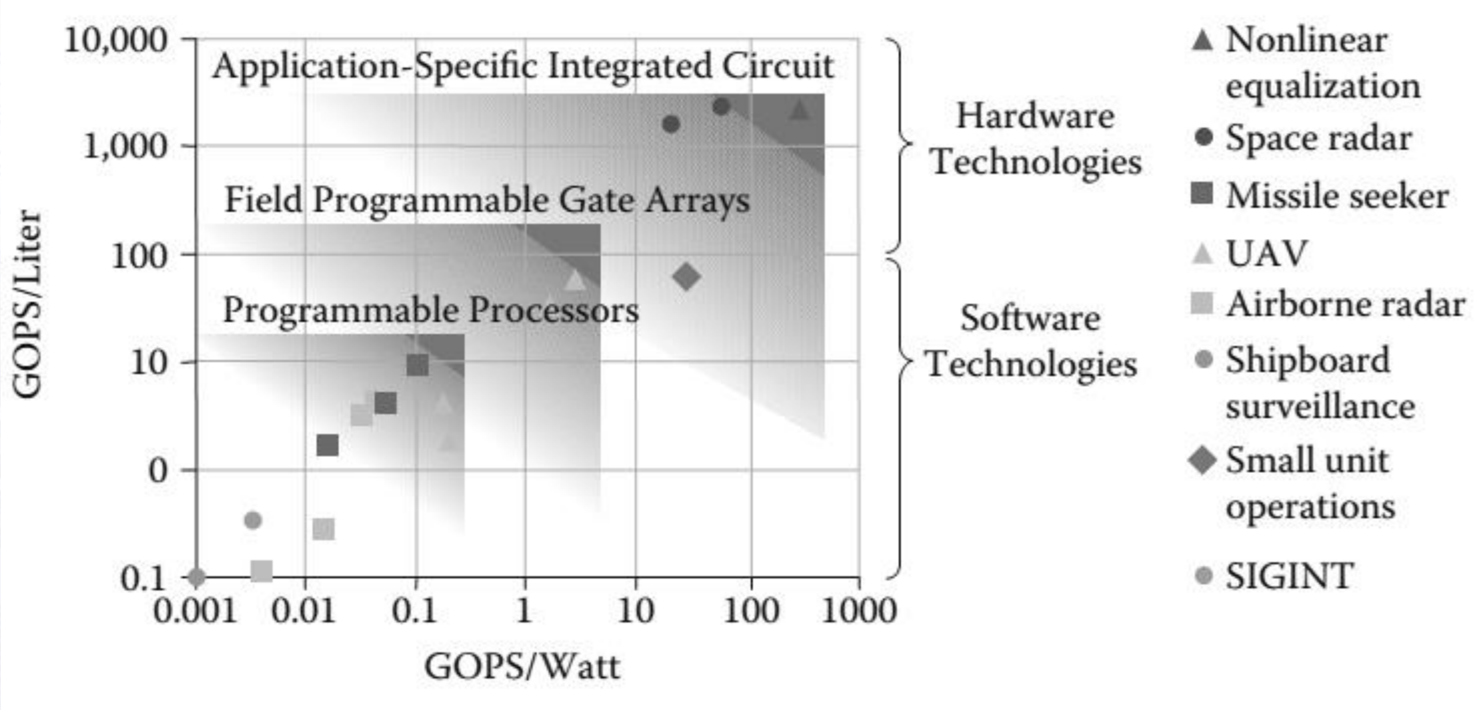
Embedded System Architecture

http://git.eti.pg.gda.pl/intel-grant/pliki/esa/Embedded_Systems_Architecture_2016_P1.pdf

KATEDRA INŻYNIERII KOMPUTEROWEJ

intel

Sponsor specjalności



Progress between 1950-2000: from 20 KOPS to 480 GFLOPS

Martinez, D. R., Bond, R. A., & Vai, M. M. (Eds.). (2008). High performance embedded computing handbook: A systems perspective. CRC Press.

Introduction

- Internet of Things (IoT)

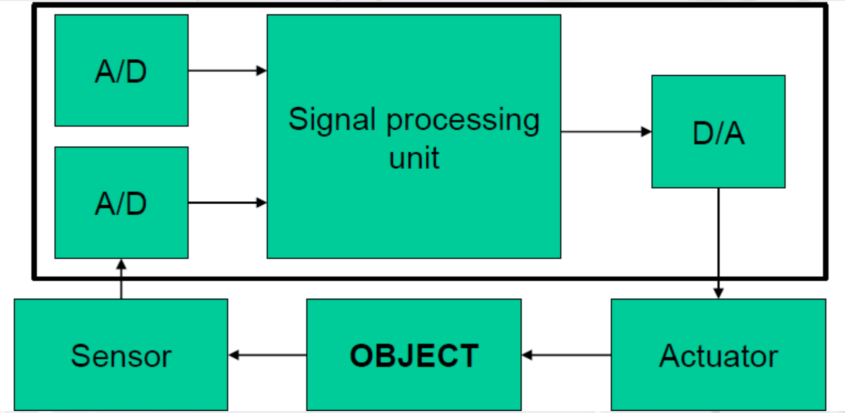
The network of physical objects or "things" embedded with electronics, software, sensors and connectivity to enable data exchange between the identified devices.

- Sensors and other elements:

Buzzer, Button, LED, Rotary Angle, Sound Sensor, Smart Relay, Temperature, Touch Sensor, Light Sensor, Mini Servo, LCD RGB Backlight

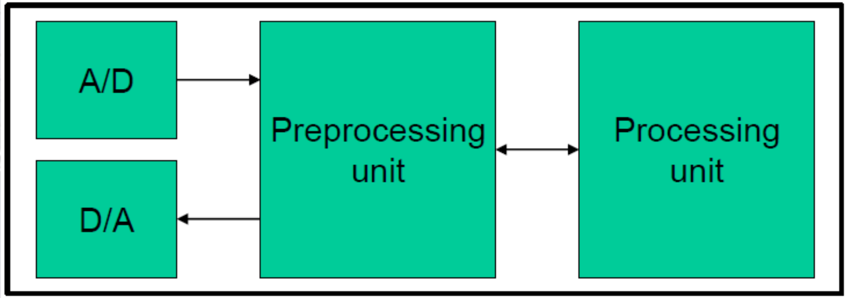
Embedded system

- A/D and D/A converters
- Sensors
- Actuators
- Signal processing unit



Embedded system

- Signal processing can requires fast preprocessing in additional unit, dedicated to that task



Introduction

Embedded system requires:

- Hardware (dedicated sensors/actuators, CPU)
- Software (operating system, drivers, algorithms, etc.)

Hardware can comprise of ready electronic boards with necessary sensors/actuators.

Software requires initialization procedure, tasks scheduling, multi-tasking for parallel events, has to solve constraints of CPU resources Software means:

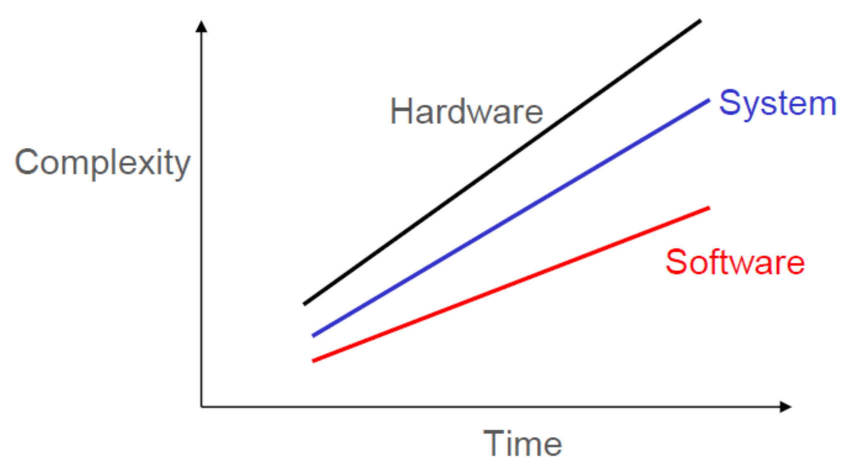
- application
- operating system (prepared mainly in C language; small percentage in assembler)

Introduction

There is a gap between hardware design, system design and software design.

Influence of economy – the way of selecting hardware.

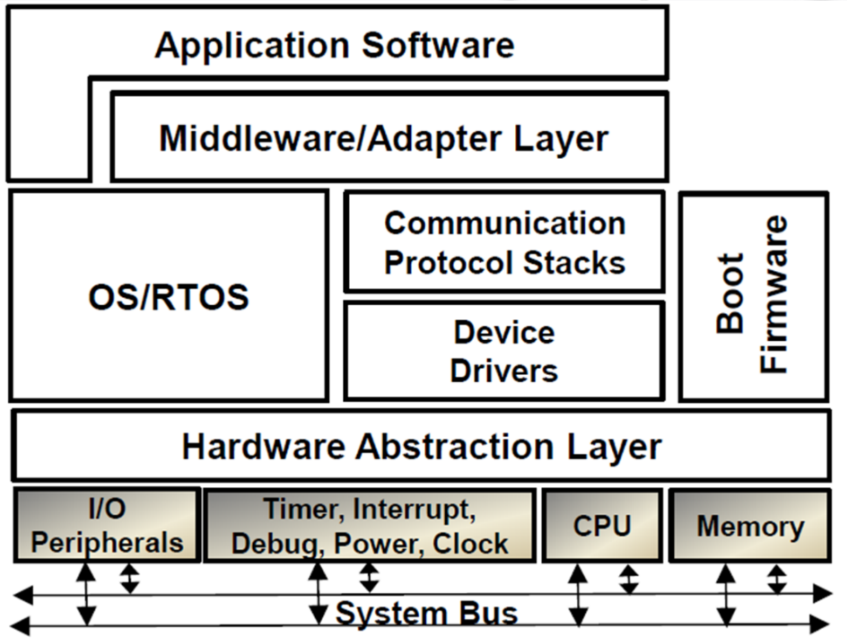
- Hardware: 2x/18 months
- System: 1.6x/18 months
- Software: 2x/24 months



ES software model

Hardware-dependent software

Ecker, W., Müller, W., & Dömer, R. (2009). Hardware-dependent Software. Springer Netherlands.

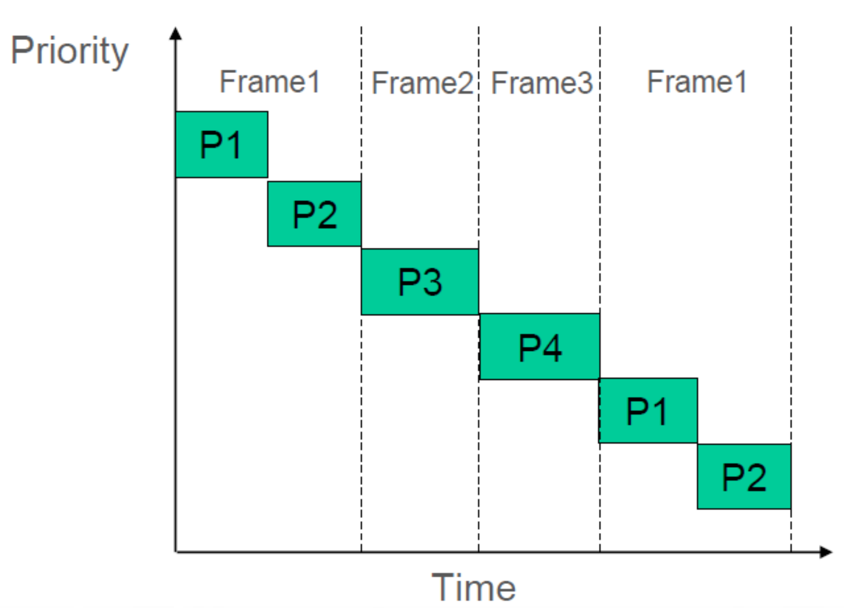


ES software model

- Application software: responsible for overall software functionality
- Middleware: provides application-specific services (e.g. database access)
- Operating System: manages and coordinates application software tasks for sharing of available software and hardware resources
- Communication Protocol Stacks: software modules on top of device drivers
- Device Drivers: software access to hardware resources by a few functions (open, initialize, access, close)
- Boot Firmware: performs initial boot proces
- Hardware Abstraction Layer: an abstract interface to access hardware resources (access, register, functional shielding)

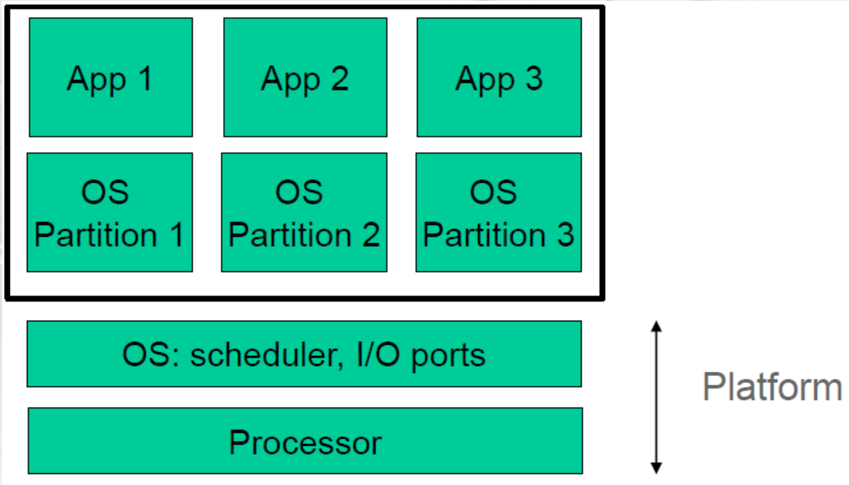
ES software model – operating system

Protection in space domain: sharing the same memory (a safety-critical application individual address spaces for processes are crucial)



ES software model – operating system

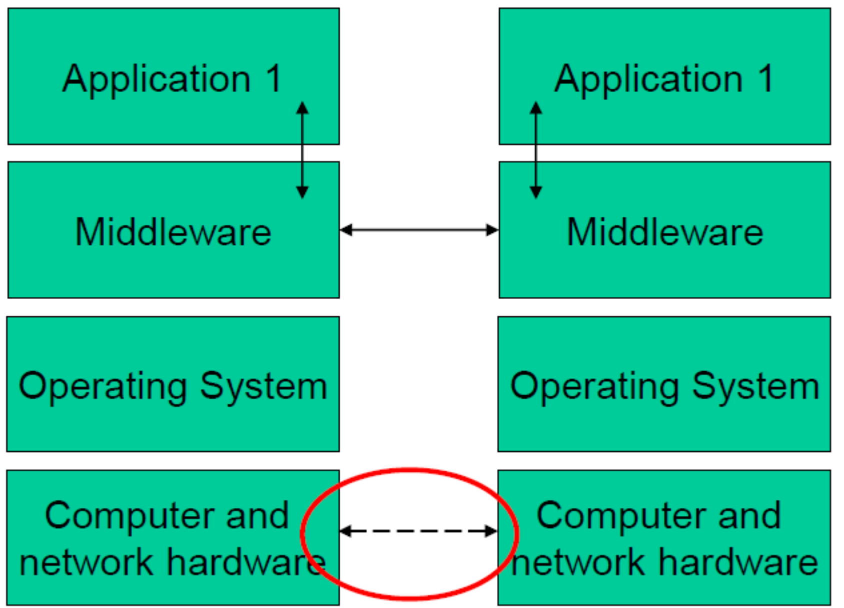
Protection in time domain: scheduler implements temporal partitioning in processes and sharing the procesor time



ES software model – communication

Communication can be performer at various layers

New systems enable tasks/data transmissions at hardware level



ES software model – initialization procedures

Initialization sequence:

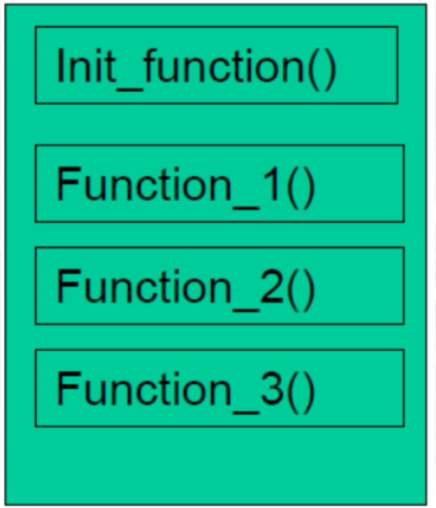
- Processor initialization
  - Intterupts disable, cache clearing
  - Data copy from ROM into RAM memories
  - Cache, interrupt vectors, system hardware initializations
  - Zeroing memory

- OS loading/booting
  - Multitasking environment initialization, interrupt and root stacks creation
  - Initialization of I/O ports, drivers, setup networks

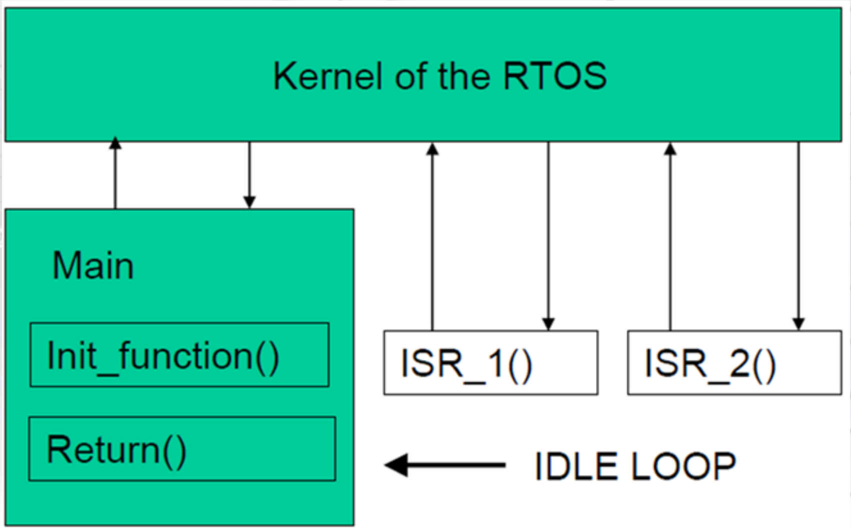
RTOS and Multi-tasking: tasks divisions and scheduling, Interrupts services

ES software structure

Application in linear programming is a set of consecutive functions

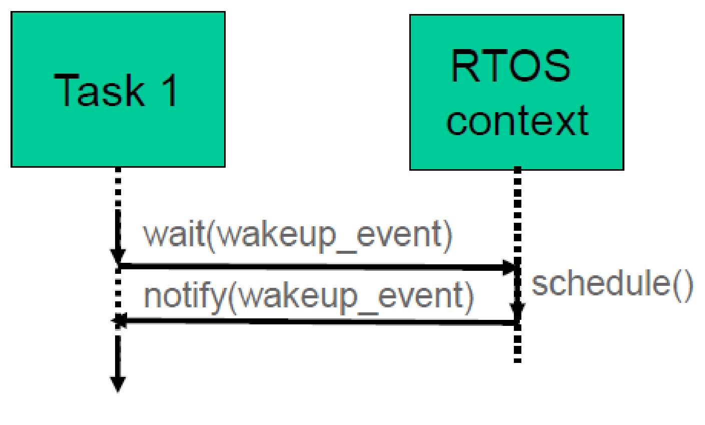


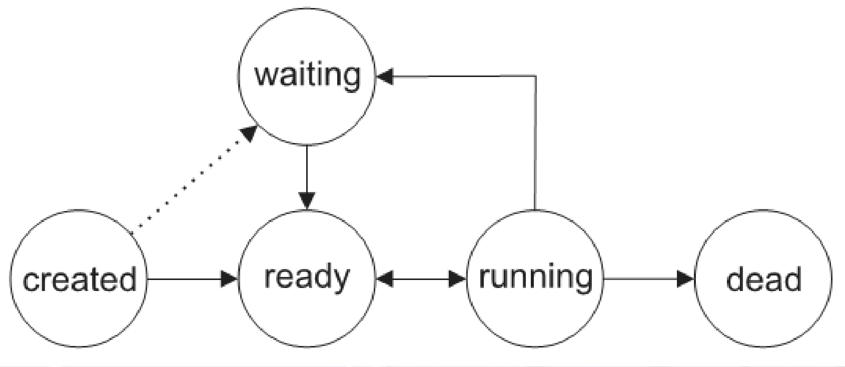
In real-time system we require the kernel to assure fast interrupt service routines



ES software model – initialization procedures

RTOS and Multi-tasking: tasks divisions and scheduling, interrupts services





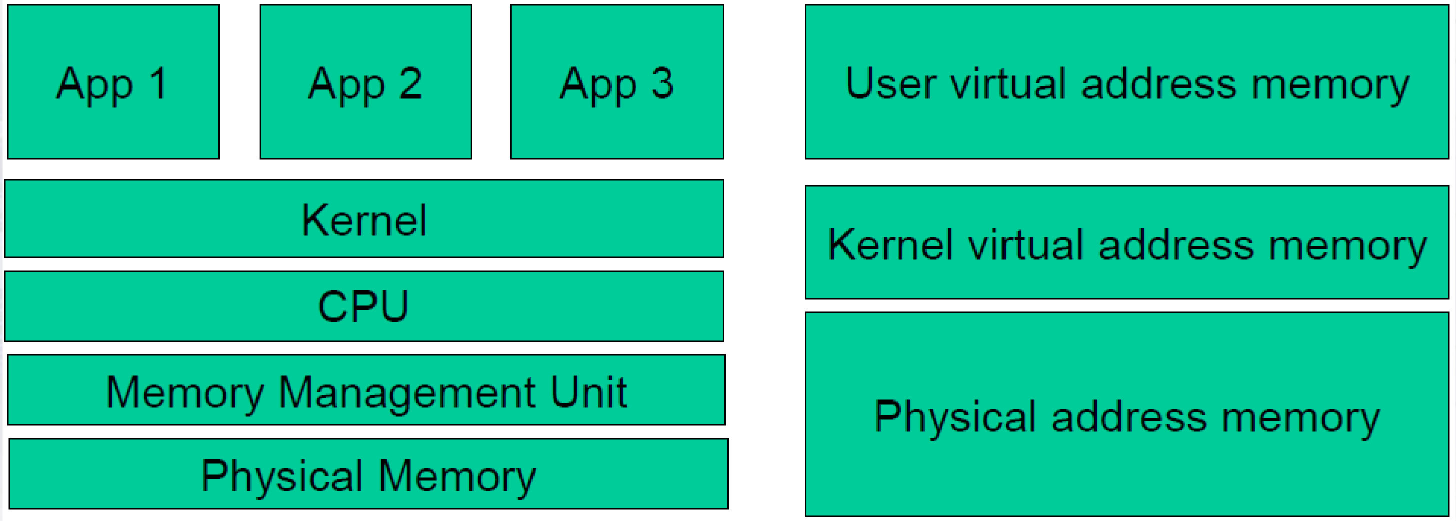
Tasks synchronization

State model of tasks in RTOS

Software task is defined by Task_start and Task_end events

ES Memory space

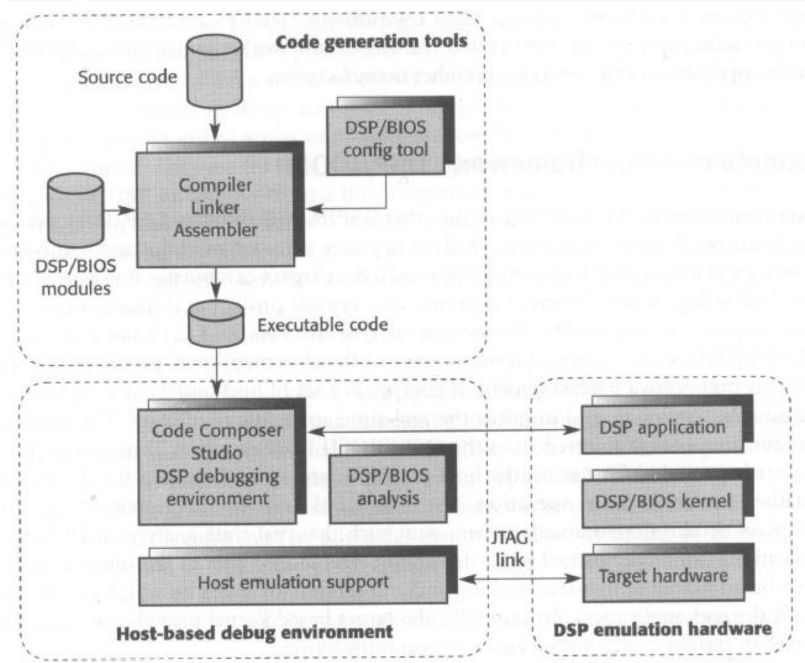
- Physical memory
- Virtual memory (user and kernel space)



KERNEL – the modules of the OS that have been loaded into RAM and have allocated memory

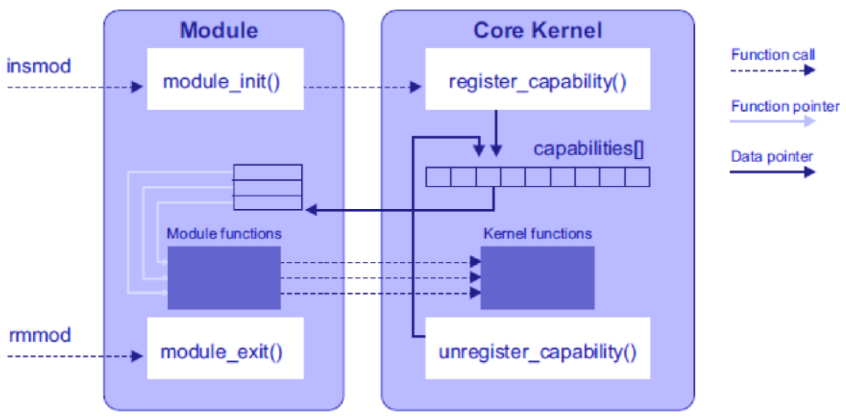
Kernel of Operating Systems and Device Drivers

- OS depends on the ES and can be established due to its functions, tasks and architecture



Kernel of Operating Systems and Device Drivers

- Kernel comprises of the modules which have to be registered in the OS
- The modules in Linux OS have names, properties and methods



KATEDRA INŻYNIERII KOMPUTEROWEJ

intel
Sponsor specjalności

Kernel of Operating Systems and Device Drivers

Device Drivers assure interface to handle requests for the device operations

Software interface to the device driver is implemented as kernel modules

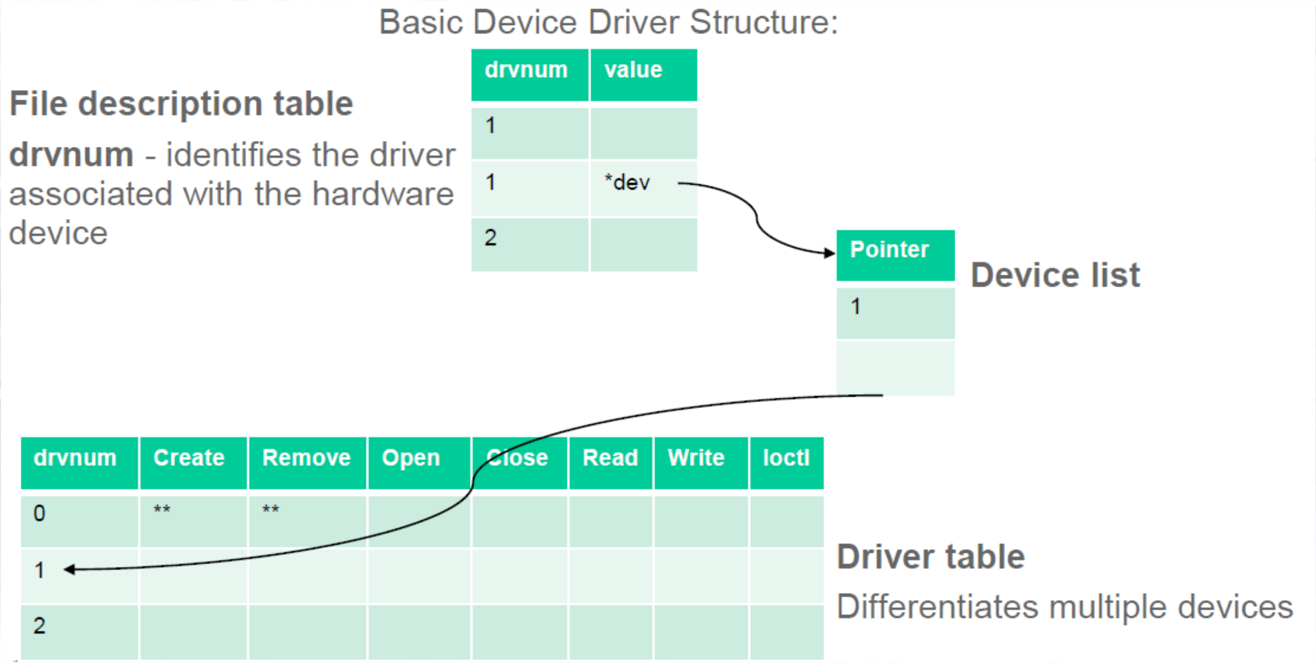
Software interface can configure device, perform I/O operations, perform interrupt requests

Devices types:

- character device (terminal) – stream of bites like a file (open, close, read, write)
- block (disk) – transfer randomly data in blocks
- network; pseudodevice

We call a device driver at initialization/configuration I/O operations, interrupts

Kernel of Operating Systems and Device Drivers



Basic Device Driver Structure:

File description table

drvnum - identifies the driver associated with the hardware device

drvnum	value
1	
1	*dev
2	

Device list

Pointer
1

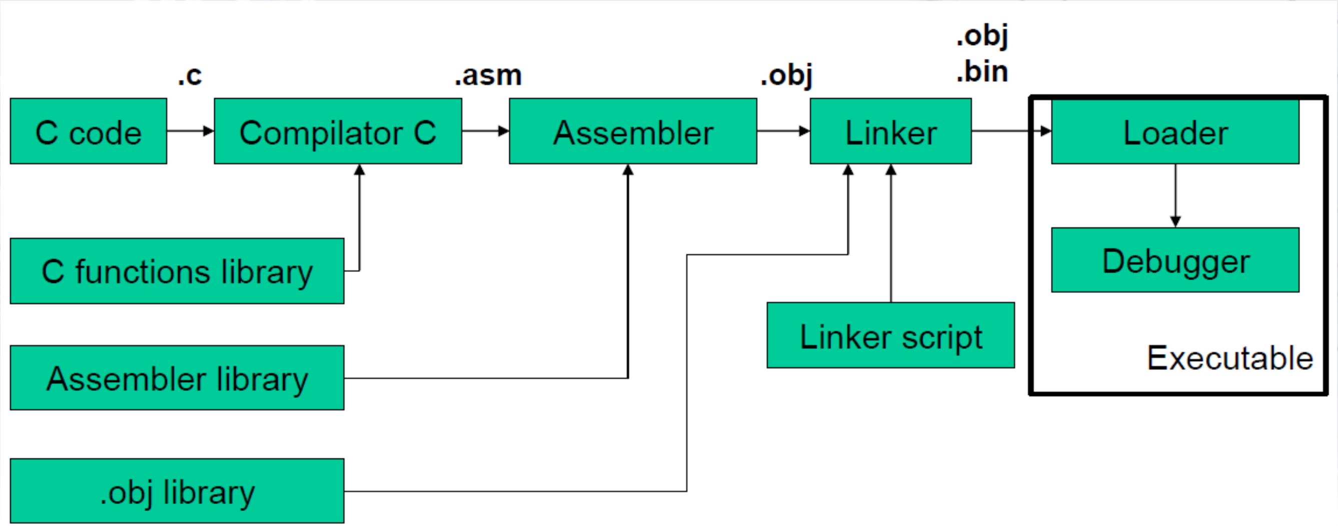
Driver table

Differentiates multiple devices

drvnum	Create	Remove	Open	Close	Read	Write	Ioctl
0	**	**					
1							
2							

ES software preparation

- The software files structure



ES compiler options

Compiler can use various options when preparing the executable code:

- Level0 – all variables are located into the registers
- Level1 – removes local variables and unused expressions + L0
- Level2 – loops optimization + L1
- Level3 – removes unused functions + L2

Programming and hardware limitations

- Detailed reports of hardware use
- Power management using sleep modes and clock rate
- Code length versus speed of the system performance
- Other programming methods

ES debugging and testing

Real-time analysis and run control:

- GUI can control: source code, registers, function calls, variables, statistics, etc.; read/write memory and registers; execution control (single step, breakpoint, watchpoint, etc.)
- Target board can communicate with the PC by JTAG, USB, Ethernet etc.
- Emulator: necessary for debugging

ES software preparation

The main rules for safety critical programming (Gerard J. Holzmann NASA/JPL Laboratory for Reliable Software Pasadena)

- Restrict to simple control flow constructs
- Give all loops a fixed upper-bound
- Do not use dynamic memory allocation after initialization
- Limit functions to no more than 60 lines of text
- Use minimally two assertions per function on average
- Declare data objects at the smallest possible level of scope
- Check the return value of non-void functions, and check the validity of function parameters
- Limit the use of the preprocessor to file inclusion and simple macros
- Limit the use of pointers. Use no more than two levels of dereferencing per expression
- Compile with all warnings enabled, and use one or more source code analyzers (e.g. available at http://spinroot.com/static/)

ES software preparation – general remarks

- **Process** – an instance of a computer program, comprises of a code and can be characterized by its current activity

Process may be made of multiple threads of executions – instructions managed independently by the scheduler (scheduler determines access to system resources)

Overheads for creating a thread are significantly less than for creating the process

Switching between the threads is much less absorbing for the OS than doing the same between the processes

ES software preparation – general remarks

- **Multitasking** – one proces serves to multiple clients

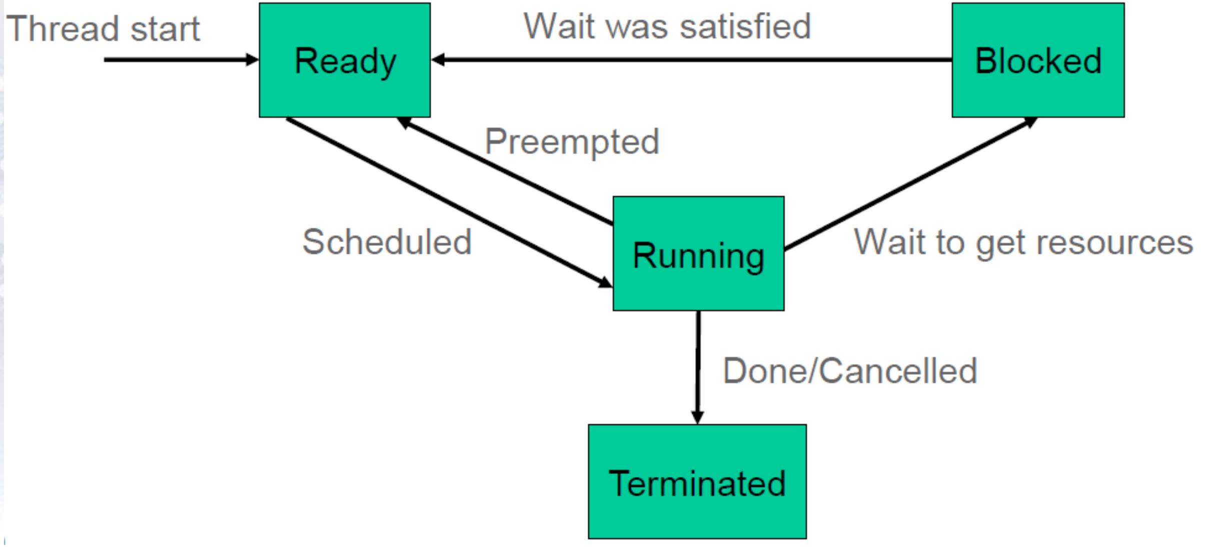
Multithreated programs have to be prepared more carefully (not all proces features are available)

Debugging is more difficult

Performance of a single procesor machine can be reduced due to multiple threads

ES software preparation – threads in Linux

- Thread states in Linux



ES software preparation

- Kernel processes in Linux
- A kernel proces can be preempted by another kernel proces
- Critical regions, for competitive resources use, can be protected by disabling interrupts or preemption
- Multiprocessor system require synchronization techniques for the data acessed by multiple CPUs
- Mutual exclusion (MUTEX):

Protection against modifying the shared data by different threads using locking/unlocking mechanism of critical sections where the data could be shared

ES software preparation

- Semaphores

A dedicated variable used for controlling access to the resources by multiple processes

It is used in parallel programming or in multi-user environment

Gets values of 0, 1 (for binary semaphores of two options locked/unlocked) or higher for arbitrary resources counting (counting semaphores)

- Semaphores can be:
  - Initialized
  - Terminated
  - Increased
  - Decreased
- Another mechanism ensuring mutual exclusion:

Busy-wait lock – lock is released and the thread waits until the lock is available

It is used when the mechanism is faster than putting the thread in a sleeping mode

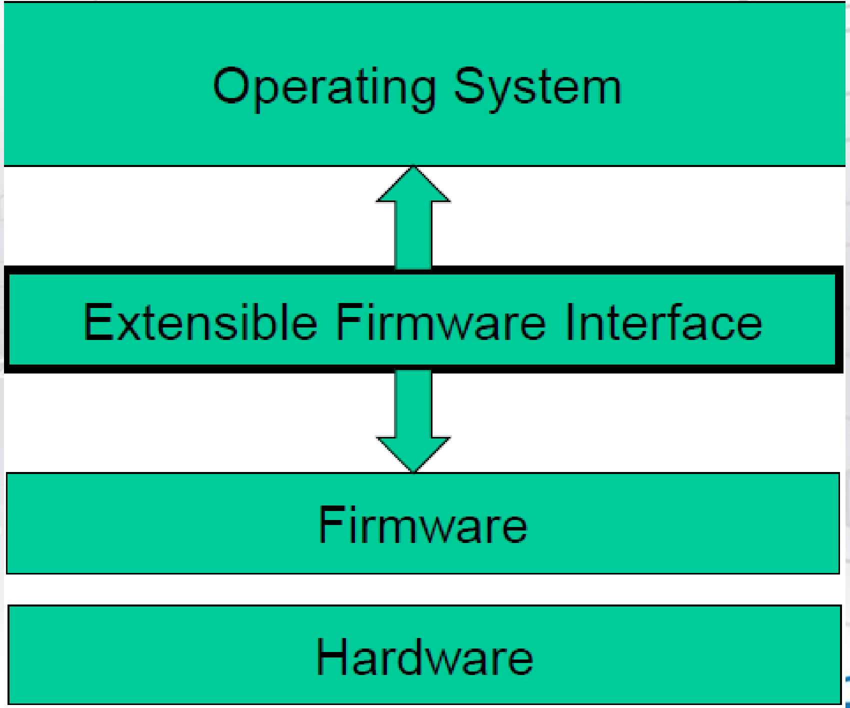
Unified Extensible Firmware Interface (UEFI)

Interface between operating system and platform firmware

- Device drivers can be prepared independently from the OS
- UEFI should replace BIOS (Basic Input/Output System)
- Will suport remote diagnostics and repair
- Is independent from the CPU
- Supports huge HDD

UEFI is answer to the limitations of the BIOS:

- 16- bits processors
- Limited HDD size



Katedra Inżynierii Komputerowej

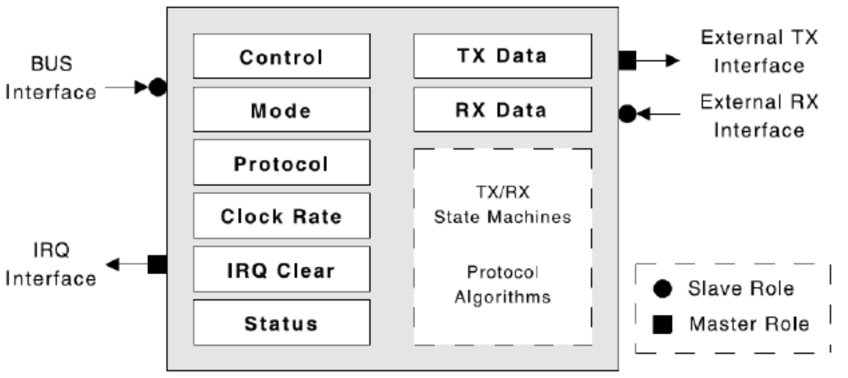
Sponsor specjalności

Hardware interfaces

Multicore systems are designed by applying various interfaces/links:

- Host Port Interface
- Serial Port (very simple and often available in the ES)
- Link Port

Clusters can be designed if the computational tasks can be divided into separate processes



Hardware interfaces

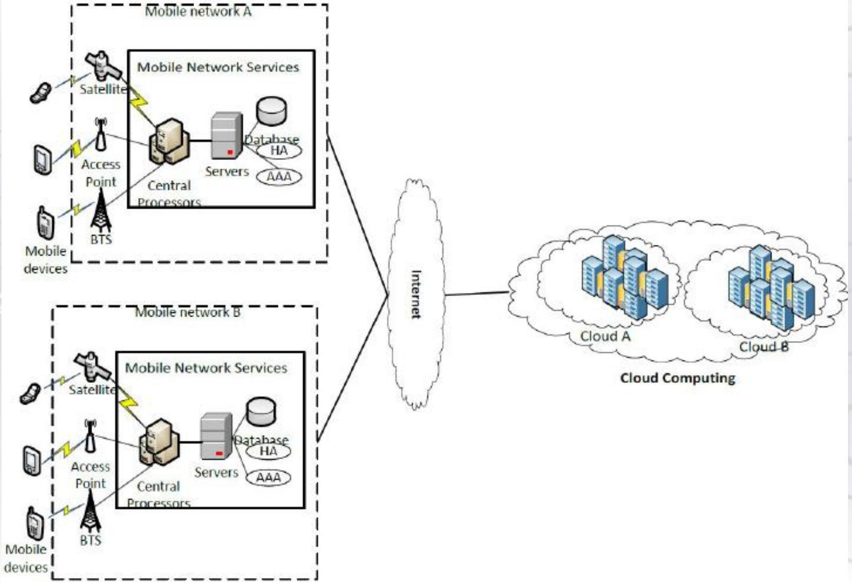
Advanced computing can be performer by cloud computing – centralized data storage and online access to computer services or resources There are pros and cons of that solution.

Pros:

- Scale and cost
- Next generation architectures
- Choice and aglity
- Encapsulated change management

Cons:

- Lock-in
- Security
- Lack of control
- Reliability



Hardware interfaces

Cloud computing.

The IoT Cloud Analytics site is provided as a service to the IoT development community

The data can be send to the cloud within a few steps only:

- Get an account from the Intel IoT Analytics Site
- Get and install the IoT Gateway Agent
  - Admin: tests connectivity, activates a device, registers time series and sends observations all from the command line;
  - Agent: runs a services by sending simple messages)
- Register your Device(s)
- Download and install the IoT Arduino Library (each sensor has to be registered)
- Write your scripts and send the data to the cloud

CPU Architecture

CPU architecture depends on type of application:

- von Neuman architecture (CPU and memory with one bus)
- Harward architecture (CPU with two buses for data and program memory separately)

Specialized chips can have combination of both types of the CPU working independently (e.g. for video coding: one for video processing and the second for controlling interfaces

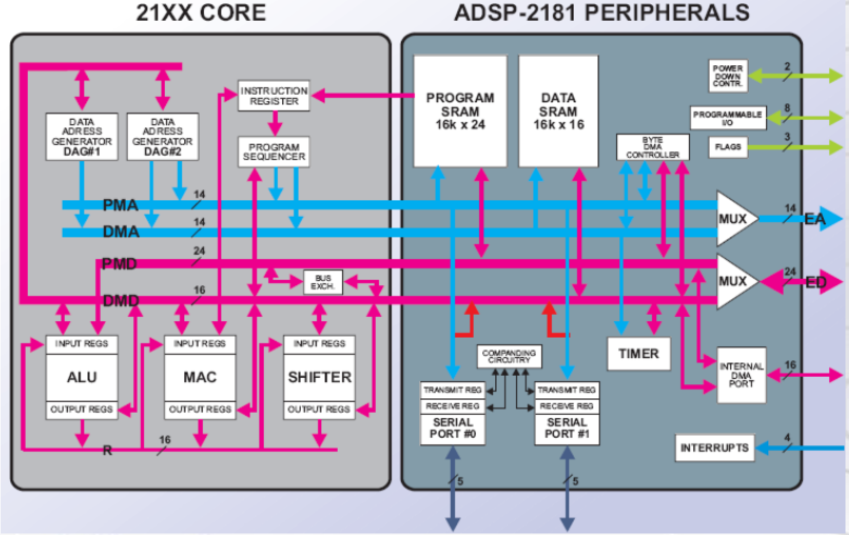
Necessary computations are performer by: ALU, MAC, SHIFTER units

CPU Architecture

CPU architecture depends on type of application

DSP optimized to perform A*B+C operations

Analog devices DSP



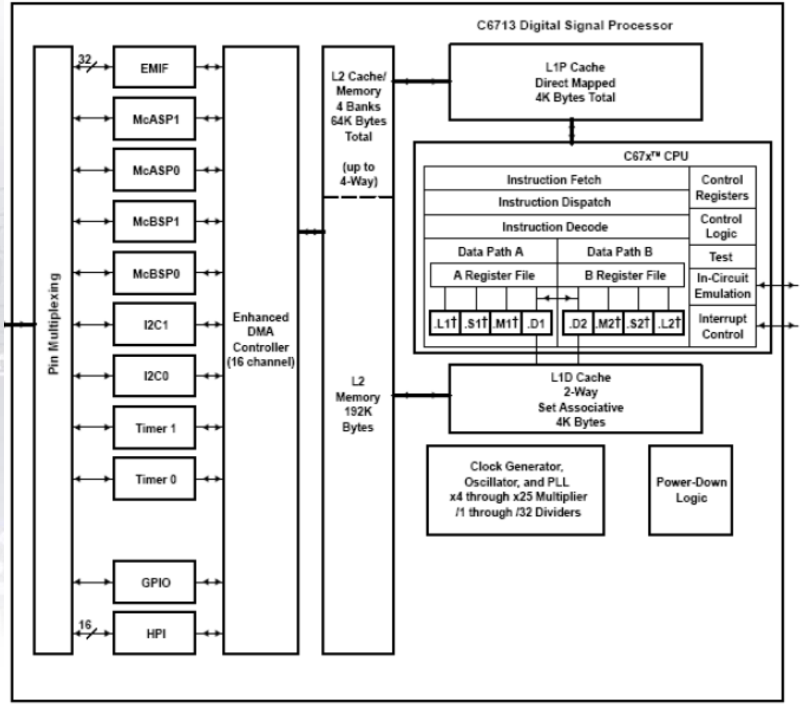
CPU Architecture

CPU architecture for DSP algorithms with two independent streams of data performing A*B+C operations

C6713 Texas Instruments Digital Signal Processor

Mechanisms:

- pipelining
- parallelism
- VLIW



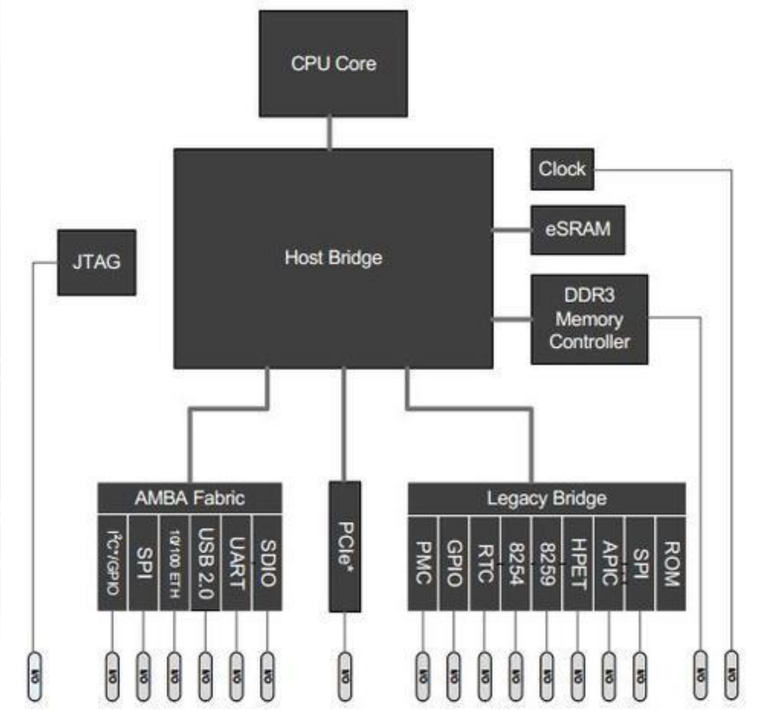
CPU Architecture

Intel Quark SoC – ultra small chips for gadgets (wearable devices – small size and low power consumption)

Cheap with developed numerous interfaces;

Applied in **Galileo board.**

The CPU instruction set is the same as a Pentium (P54C/i586) CPU



CPU Architecture

Intel Quark SoC

- 400 MHz maximum operating frequency
- Cache, internal memory (flash, SRAM),
- Low power options to run at half or at quarter of maximum CPU frequency
- 32-bit address bus, 32-bit data bus
- 16 Kbyte shared instruction and data L1 cache
- Interfaces: UART, USB Host Port, I2C, SPI

Power management:

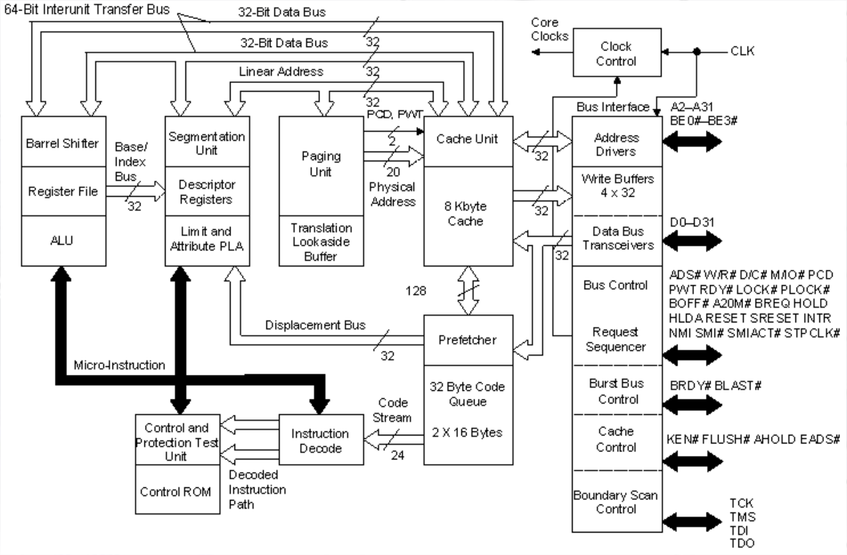
- S0 – full power on
- S1, S2, S3, S4 – sleeping states; parts of the chip are in a sleeping state to reduce power consumption
- S5 - system is switched off

- Operating system: Linux (e.g. Debian 7.0 Wheezy)
- one-fifth the size and one-tenth the power of low-end Atom chip

CPU Architecture

Intel Quark internal architecture includes:

- 32 bit RISC integer core;
- Single cycle execution;
- Instruction pipelining;
- Floating-point unit;
- Cache for data and instructions;
- Memory management unit



CPU Architecture

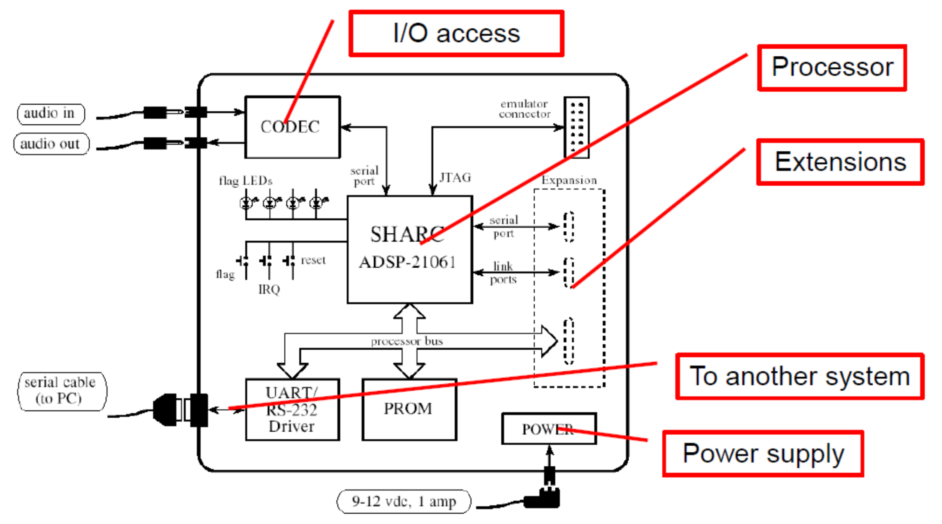
Atom Silvermont – low-power Atom processors used in systems on a chip applied in Intel Edison board for tablets, smartphones and other wearable devices; launched by Intel in 2012

Applied in the **Intel Edison board**:

- high performance with high power efficiency
- pipeline mechanisms
- multi-core support (two cores, second-level cache)
- WiFi

Development boards

Various SDK systems have been proposed for different digital technologies. The common elements are:



Development boards

- Arduino – open-source computer software and hardware for sensing and controlling physical world; designed for Atmel AVR microcontrollers and microprocessors

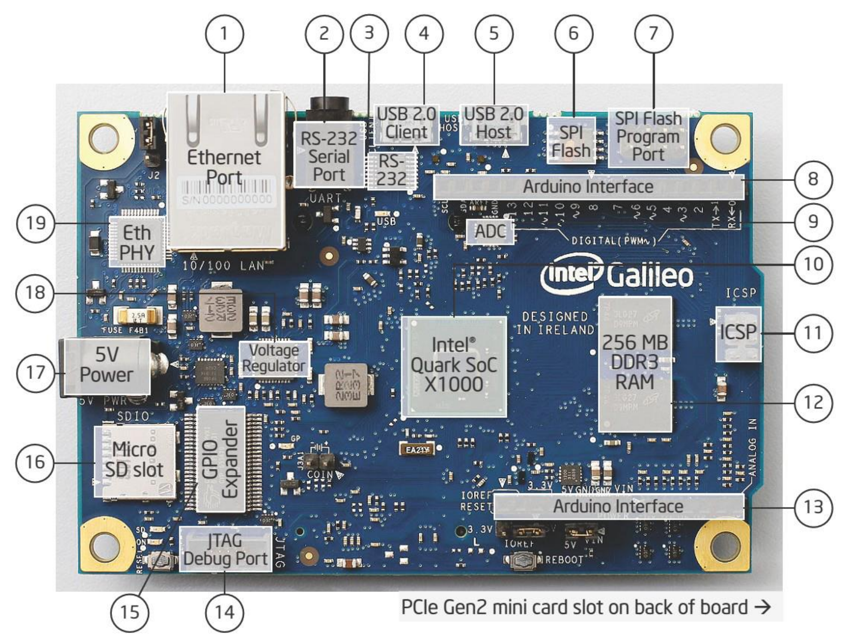
- STM32F4 DISCOVERY 32-bits ARM Cortex®-M4 processor; much faster with additional elements (build-in sensors, touchscreen, camera, memory card)

Development boards

Analog Devices Blackfin BF548 EZ-kit

- touchscreen, keyboard, HDD, additional interfaces for video and audio applications

Development boards - Galileo



400MHz 32-bit Intel® Pentium instruction set architecture (ISA)-compatible processor o 16 KBytes on-die L1 cache 512 KBytes of on-die embedded SRAM

Simple to program: Single thread, single core, constant speed

ACPI compatible CPU sleep states supported

An integrated Real Time Clock (RTC), with an optional 3V “coin cell” battery for operation between turn on cycles; 10/100 Ethernet connector

Full PCI Express mini-card slot, with PCIe 2.0 compliant features

Works with half mini-PCIe cards with optional converter plate

Provides USB 2.0 Host Port at mini-PCIe connector

USB 2.0 Host connector

Support up to 128 USB end point devices

USB Device connector, used for programming

10-pin Standard JTAG header for debugging

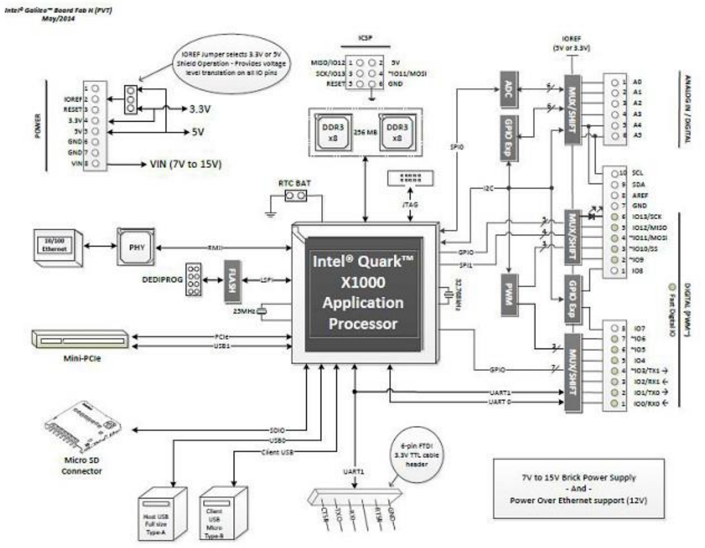
Reboot button to reboot the processor

Reset button to reset the sketch and any attached shields

Katedra Inżynierii Komputerowej

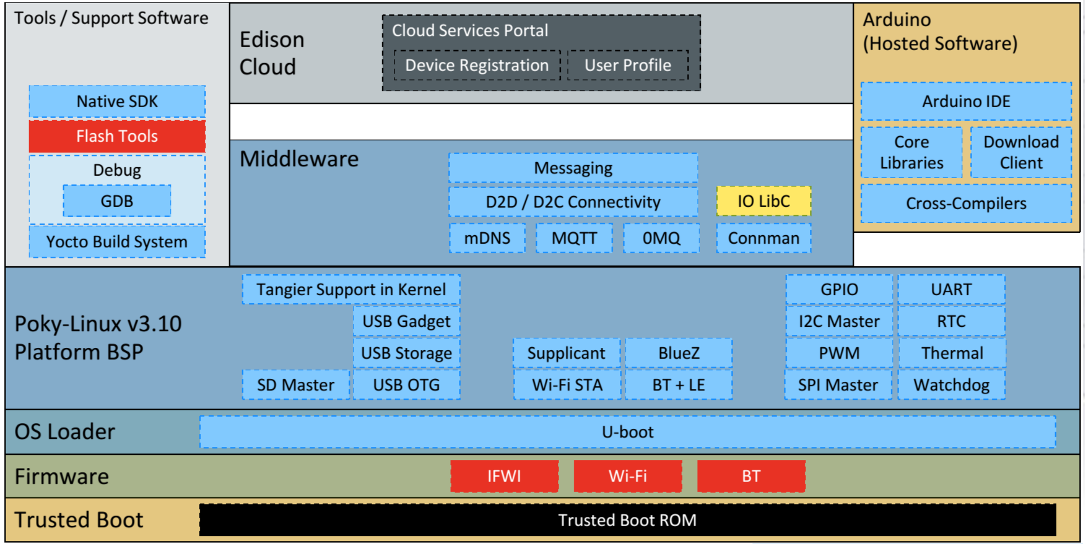
Sponsor specjalności

Development boards - Galileo



Development boards – software

- Intel Edison: software stack



Tools / Support Software

- Native SDK
- Flash Tools
- Debug
  - GDB
- Yocto Build System

Edison Cloud

- Cloud Services Portal
  - Device Registration
  - User Profile

Arduino (Hosted Software)

- Arduino IDE
- Core Libraries
- Download Client
- Cross-Compilers

Middleware

- Messaging
- D2D / D2C Connectivity
- mDNS
- MQTT
- 0MQ
- IO LibC
- Connman

Poky-Linux v3.10 Platform BSP

- Tangier Support in Kernel
- USB Gadget
- USB Storage
- SD Master
- USB OTG
- Supplicant
- BlueZ
- Wi-Fi STA
- BT + LE
- GPIO
- UART
- I2C Master
- RTC
- PWM
- Thermal
- SPI Master
- Watchdog

OS Loader

- U-boot

Firmware

- IFWI
- Wi-Fi
- BT

Trusted Boot

- Trusted Boot ROM

Development boards – fast programming

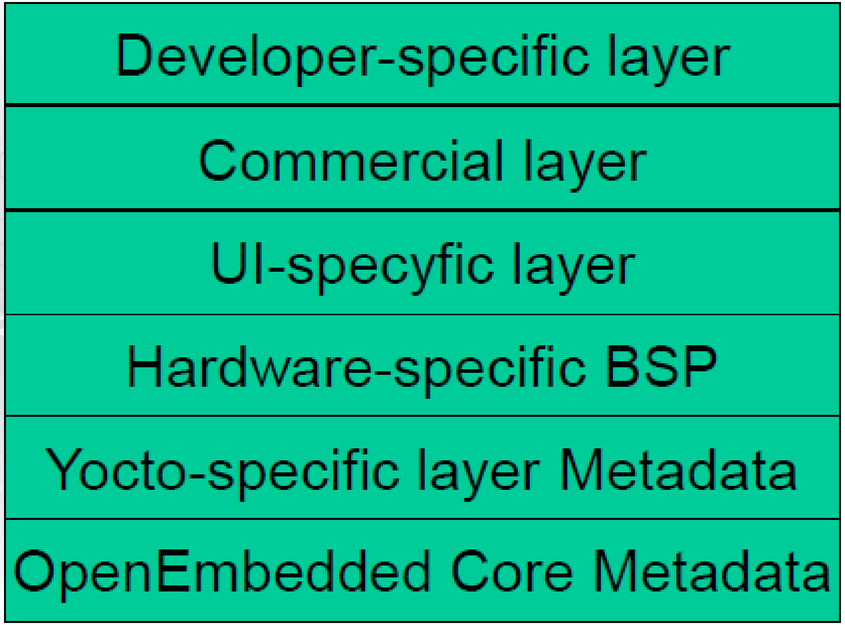
Additional tools help to accelerate fast programming and apps preparation

- Integrated Performances Primitives
  - Signal processing
  - Image processing
  - Matrix operations
  - Cryptograph
- Math Kernel Library
  - Various Math functions (e.g. FFT)
- Cordova
  - Building apps for BlackBerry, IOS, Android, Windows desktop
- Sensors
  - Sensors fusion (various measurements at the same time improve applications and results)

Development boards – system preparation

Intel Edison Yocto Project:

- Open source project aimed to limit time for making the build systems
- Compatible with various linux distributions (Ubuntu, Fedora, openSUSE, CentOS, Debian); stable releases are issued within a few months
- Helps to develop industrial standards of reliable operating systems for ES
- Software comprises of separate layers (each layer can be prepared using various set of files)



Developer-specific layer

Commercial layer

UI-specyfic layer

Hardware-specific BSP

Yocto-specific layer Metadata

OpenEmbedded Core Metadata