Introduction to Computer Interfacing and Embedded Systems

What is ECE 315 all about?

- ECE 315 is titled “Computer Interfacing” but the course actually covers material from a broader range of areas:
  - ❖ Microcomputers and hardware interfaces
  - ❖ Software for real-time embedded systems
  - ❖ Computer communications interfaces

- This course considers the design and debugging of systems that involve the interaction of microcomputer ***hardware***, embedded ***software***, and ***communications interfaces***.

- ECE 315 is intended to prepare students for ECE 492, the Computer Engineering Design Project.

Prerequisites for ECE 315

1) A first course in ***microcomputer architecture***, such as:
   ECE 212 – Introduction to Microprocessors, *or*
   ECE 311 – Computer Organization and Architecture, *or*
   CMPUT 229 – Computer Organization and Arch. I, *or*
   Permission of the instructor

2) An intermediate course in ***C/C++ programming***, such as:
   CMPUT 201 – Practical Programming Methodology, *or*
   ECE 220 – Programming for Electrical Engineering, *or*
   Permission of the instructor

Other courses that are related to ECE 315

1) A survey course in ***operating systems***, such as:
   CMPUT 379 – Operating Systems Concepts

2) A survey course in ***computer communications***, such as:
   CMPUT 313 – Computer Networks, *or*
   ECE 487 – Data Communications Networks

The key required concepts from operating systems and computer communications will be covered in ECE 315.

What is an “Embedded System”?

- ***Personal Computers*** (PCs) are the most obvious form of programmable digital computer. There is a screen for outputting text and images to a human user, and a keyboard and mouse (or touch-screen equivalents) for inputting characters and selection decisions from menus.
- However, the number of PCs and tablets in an advanced economy is dwarfed by at least a factor of 10 by the much larger number of embedded systems.
- An ***embedded system*** is a computer system that is used to monitor and control a product or engineering system. An embedded system often has only a simplified and/or application-specific user interface. A conventional PC or tablet user interface might be absent in such a system.

Industrial Process Control

- Process control systems are widely used to allow a relatively small number of human operators to control a complex industrial process (e.g, oil refinery, natural gas processing plant, pulp mills, power generation plant, car assembly line) with high consistency, efficiency, and safety.
- A wide variety of technologies are used to implement modern *process control systems* (PCSs):
  - Ladder logic*, Programmable Logic Controllers* (PLCs)
  - *Remote Terminal (or Telemetry or Telecontrol) Units* (RTUs)
  - *Proportional-integral-derivative* (PID) controllers
  - *Supervisory Control and Data Acquisition* (SCADA) systems
  - *Distributed Control Systems* (DCSs)
  - The “Internet of Things”, Edge (or Fog, or Mist) Computing
- The focus in this course will be on software-programmed, networked embedded microcomputers, which are widely used in modern PCSs.

Ex: Allen-Bradley Micro850 PLC

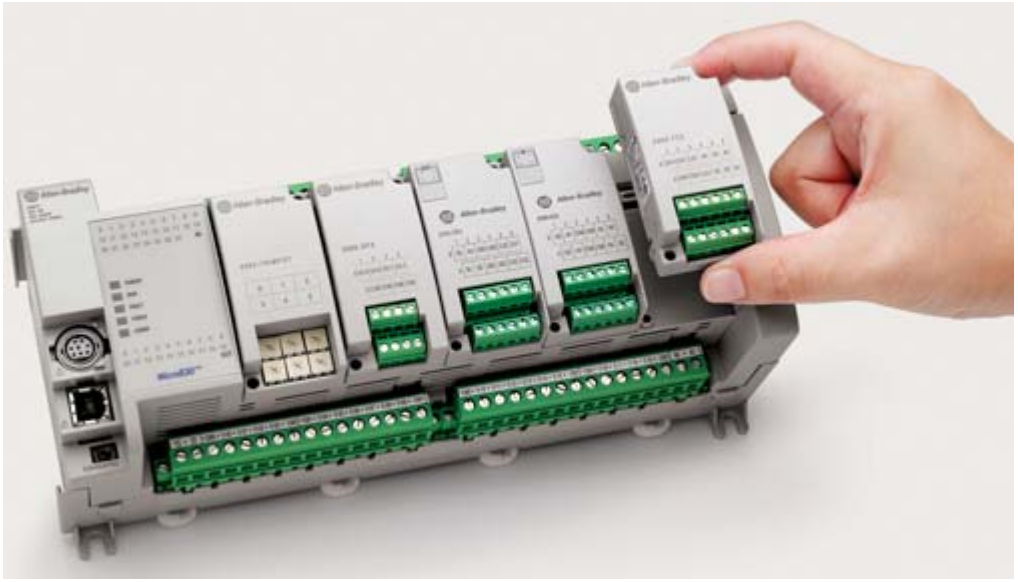


Photo courtesy of Rockwell Automation

- A typical high-end *Programmable Logic Controller* (PLC)
- Highly customizable with expansion I/O modules, terminal blocks, etc.
- Can be expanded to handle up to 132 digital input/output signals
- Can be programmed using three standard PLC methods: (1) ladder diagrams, (2) function blocks, and (3) structured text.

Supervisory Control and Data Acquisition Systems

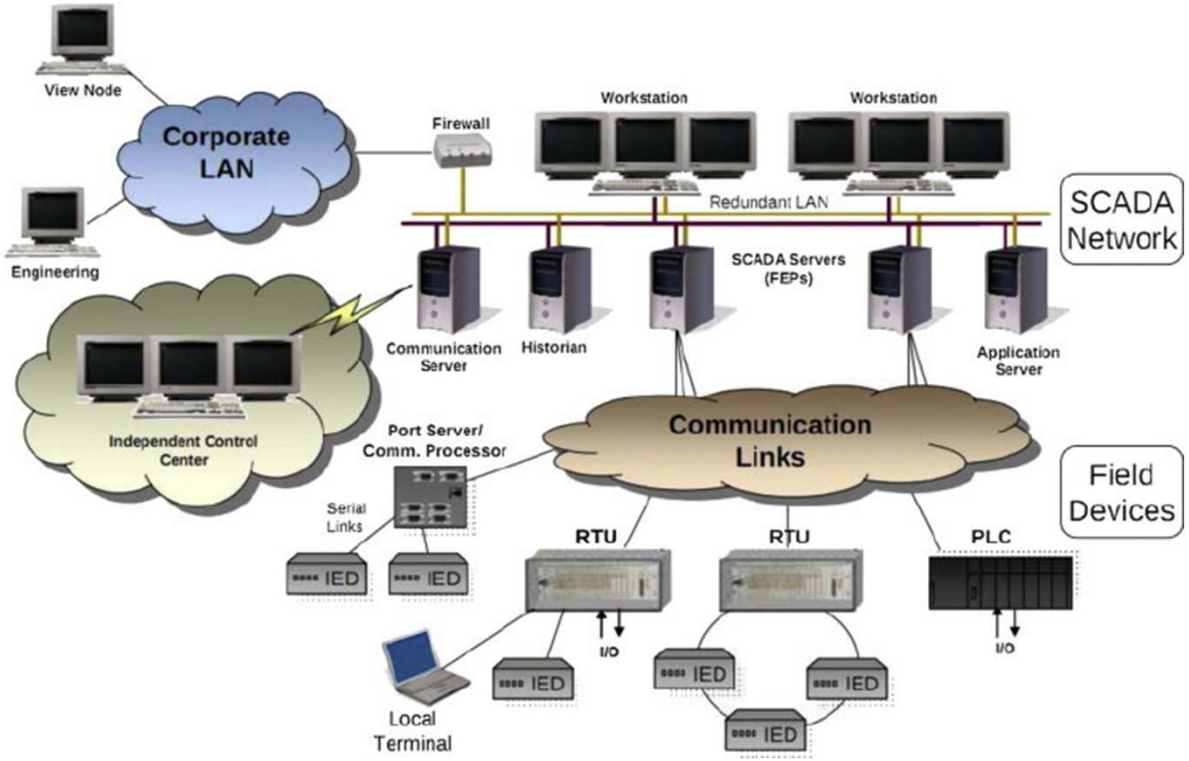


Photo courtesy of Dr. Helge Janicke, De Montfort Univ.

- SCADA systems are typically geographically distributed systems that coordinate and supervise *remote terminal units* (RTUs) and *programmable logic controllers* (PLCs) over a communications network.
- SCADA systems play a major role in Alberta industry (e.g., oil & gas)

The Internet of Things (IoT)

- "The ***Internet of Things*** (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and network connectivity which enables these objects to connect and exchange data." [Wikipedia, Jan. 8, 2017]
- The term was reportedly introduced by Kevin Ashton in a presentation that he gave at Proctor & Gamble in 1999.
- There is nothing particularly new about the Internet of Things: at present it is a fashionable term that describes a technology that has been emerging for many years already.
- The current popularity of the Internet of Things is serving a useful purpose in that it highlights both the growing *opportunities* as well as the *risks* of the increasingly pervasive use of networked embedded systems.

Other embedded computing technologies

There is a somewhat confusing variety of terminologies:

- Sensor networks, mechatronics
- The Internet of Things, smart distributed infrastructure
- Cloud computing vs. edge computing
- Fog computing, mist computing
- Ubiquitous computing (ubicomp), pervasive computing
- Smartphone, smart city, smart grid, smart home, etc.
- Ambient Intelligence

New terms come into fashion all the time.

The Microcomputer "Brain"

- Lying at the heart of an embedded system is a microcomputer (μC) that contains a ***Central Processing Unit*** (CPU), different kinds of ***memory***, and ***input/output subsystems***. The CPU fetches, decodes & executes software instructions.
- A ***microprocessor*** (μP) is a digital system that contains a CPU and closely related subsystems such as an ***interrupt handling subsystem*** and a ***cache memory***.
- A ***microcontroller unit*** (MCU) is a miniaturized system-on-a-chip (SoC) that contains, on a single semiconductor integrated circuit (IC), one (or more) ***microprocessor(s)*** and ***supporting subsystems*** (e.g., timers, interfaces) that are programmed to implement the desired embedded system.

Where do we find Embedded Systems?

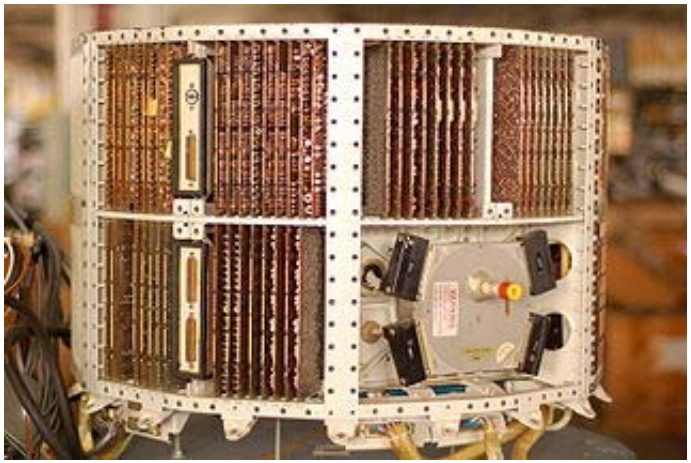
- In ***consumer electronics***: TVs, cell phones, electronic games, WiFi routers, entertainment systems, cameras, ...
- In ***residences***: microwave ovens, refrigerators, thermostats, high-efficiency furnaces, security systems, solar panels, ...
- In ***automobiles***: used to control the ignition timing, the fuel injectors, electrical power generation and distribution, fault detection and diagnosis, “black box” event recording, anti-lock braking, entertainment systems, anti-theft, wireless locking & ignition, maps & navigation, autonomous driving, ...
- In ***industry***: infrastructure (e.g., communications, electrical power, banking system, stock markets), numerical controlled tools, factories, sawmills, refineries, oil fields, gas plants, ...

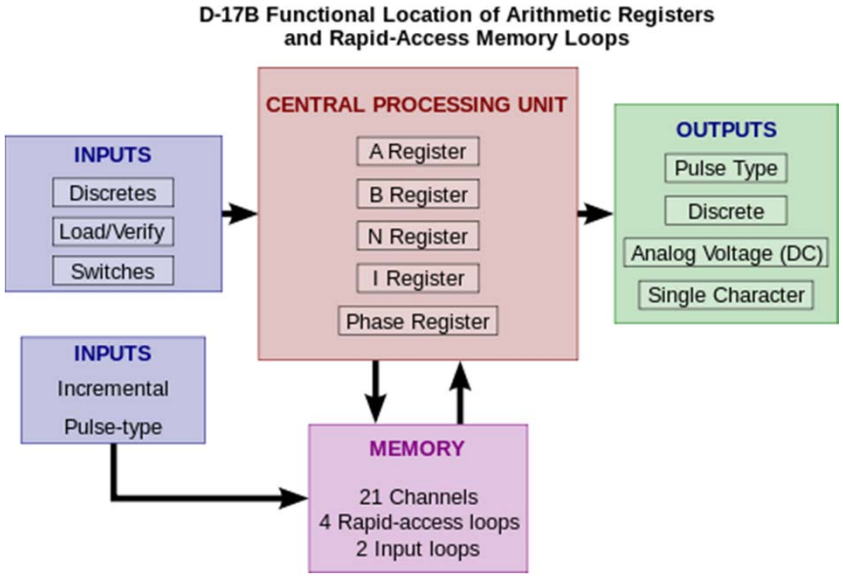
Why Use Embedded Systems?

- Embedded systems provide high-speed, ***software-programmable*** sensor monitoring and signal processing, data processing, actuator control, communications capabilities, etc.
- Computerization using an embedded system permits the use of ***sophisticated control algorithms*** that can provide greater efficiency, productivity, reliability, flexibility, upgradeability, safety, and profitability (both for the vendors and users).
- Embedded systems can compensate for the inaccuracies of lower-quality sensors and actuators to ***provide an effectively higher-quality system at a lower total cost***.
- Hardware and software companies continue to push for new applications of embedded systems to grow their markets.

Early Embedded Systems (1)

- 1962: Autonetics / North American Aviation D-17B military computer used to implement the guidance system for the Minuteman I ICBM. Weighing 28 kg, the D-17B contained 1521 discrete transistors, 6280 diodes, 1116 capacitors, 504 resistors, and dissipated 250 W at a clock frequency of 345.6 Hz.



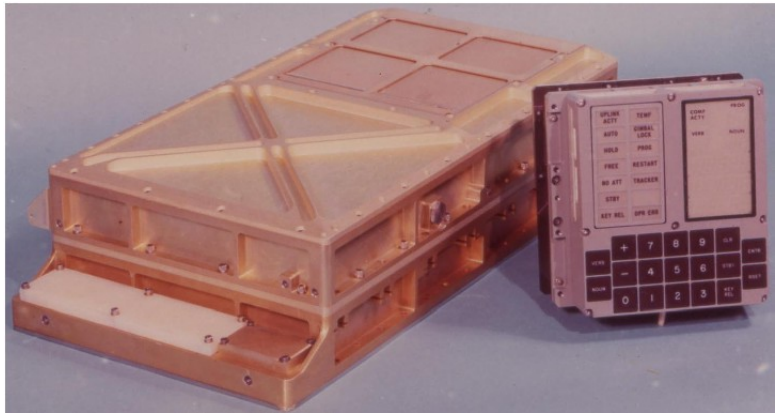


Data and images from www.wikipedia.org

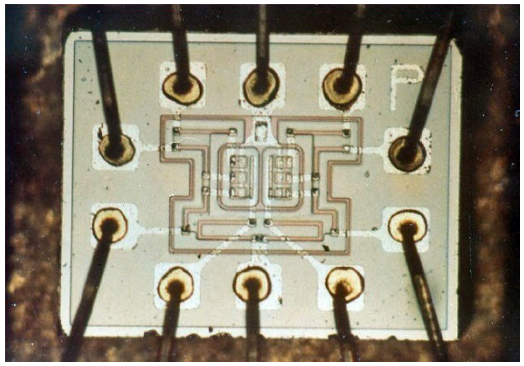
Early Embedded Systems (2)

- 1966: MIT Instrumentation Laboratory / Raytheon Apollo Guidance Computer (AGC), used to control the Apollo spacecraft. Weighing 32 kg, the AGC contained 2800 integrated circuits (dual 3-input NOR gates) and dissipated 55 W with a 4-phase 1.024-MHz clock.

Apollo Command & Service Modules



AGC Computer and DSKY User Interface



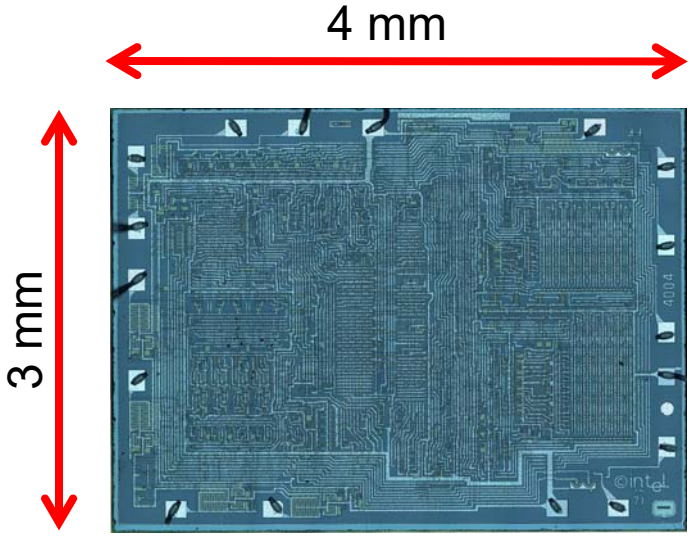
RTL Dual 3-input NOR IC

Data and images from www.wikipedia.org

Early Embedded Systems (3)

- 1970: The Busicom 141-PF calculator was designed, in collaboration with Intel Corp. to use a 4-bit microprocessor, the 4004. The 4004 was the first commercially successful microprocessor IC. To implement the full embedded system, the 4004 was supported with a 256-byte ROM & 4-bit I/O port (the 4001), a 40-byte RAM (the 4002) and a 10-bit parallel shift register (the 4003). The 4004 contained 2300 transistors and used a 740 kHz clock to execute from 46300 to 92600 instructions per second.

Unicom 141P, a version of the Busicom 141-P



Intel 4004 µP in 10-µm PMOS

Data and images from www.wikipedia.org

The Irving Oil Refinery in St John, NB

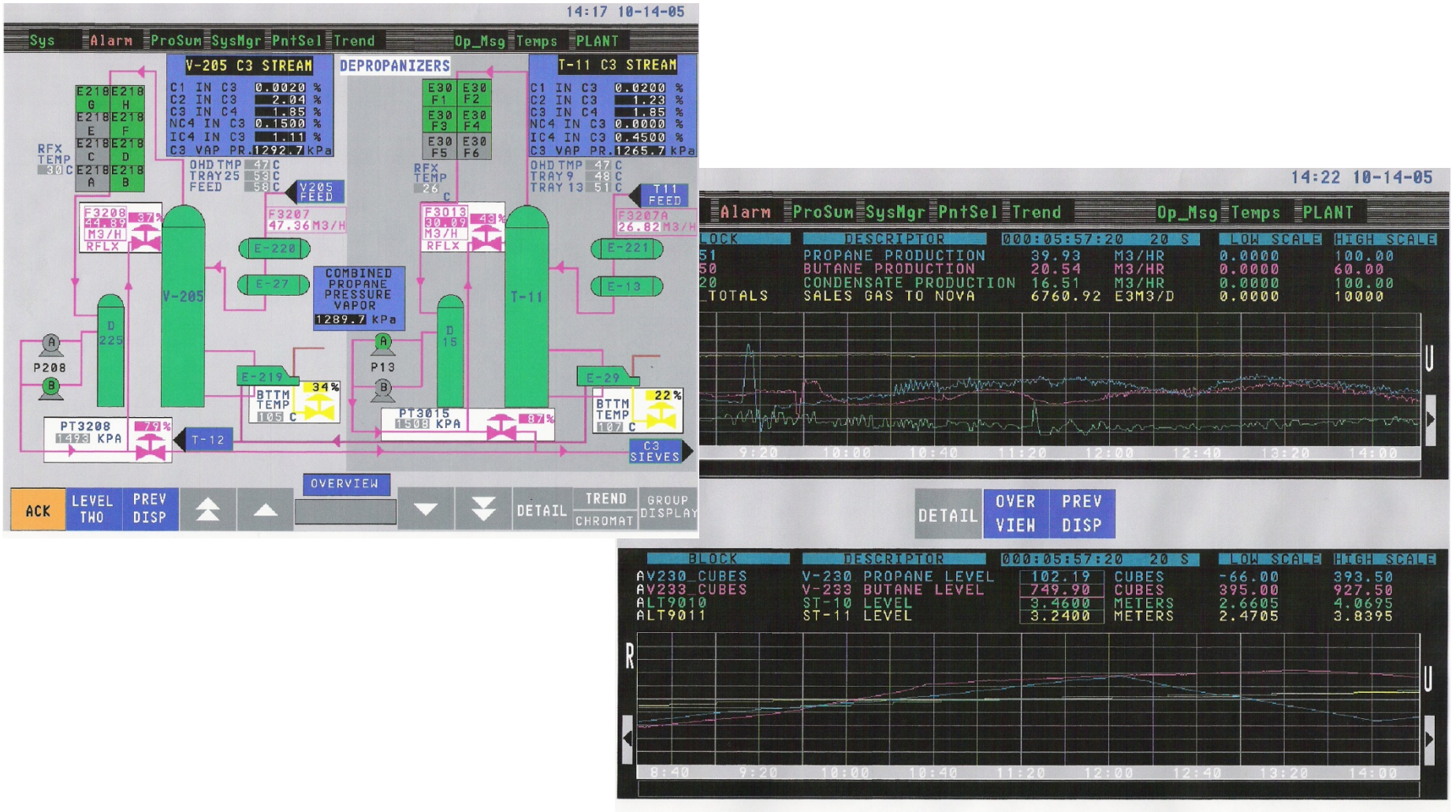
- Canada's largest oil refinery, capable of processing 300,000 barrels of oil per day
- Computer control is required to maximize production efficiency and ensure safety.

Photos from the Globe and Mail

Keyera's Gas Complex in Rimbey, AB

Photo courtesy of Keyera Corp.

Views of a Control Console at the Gas Plant



Photos courtesy of Keyera Corp.

Apple iPhone 7 Teardown

Photos from TechInsights (Ottawa, ON)

Apple iPhone 7 Main Printed Circuit Board (PCB)

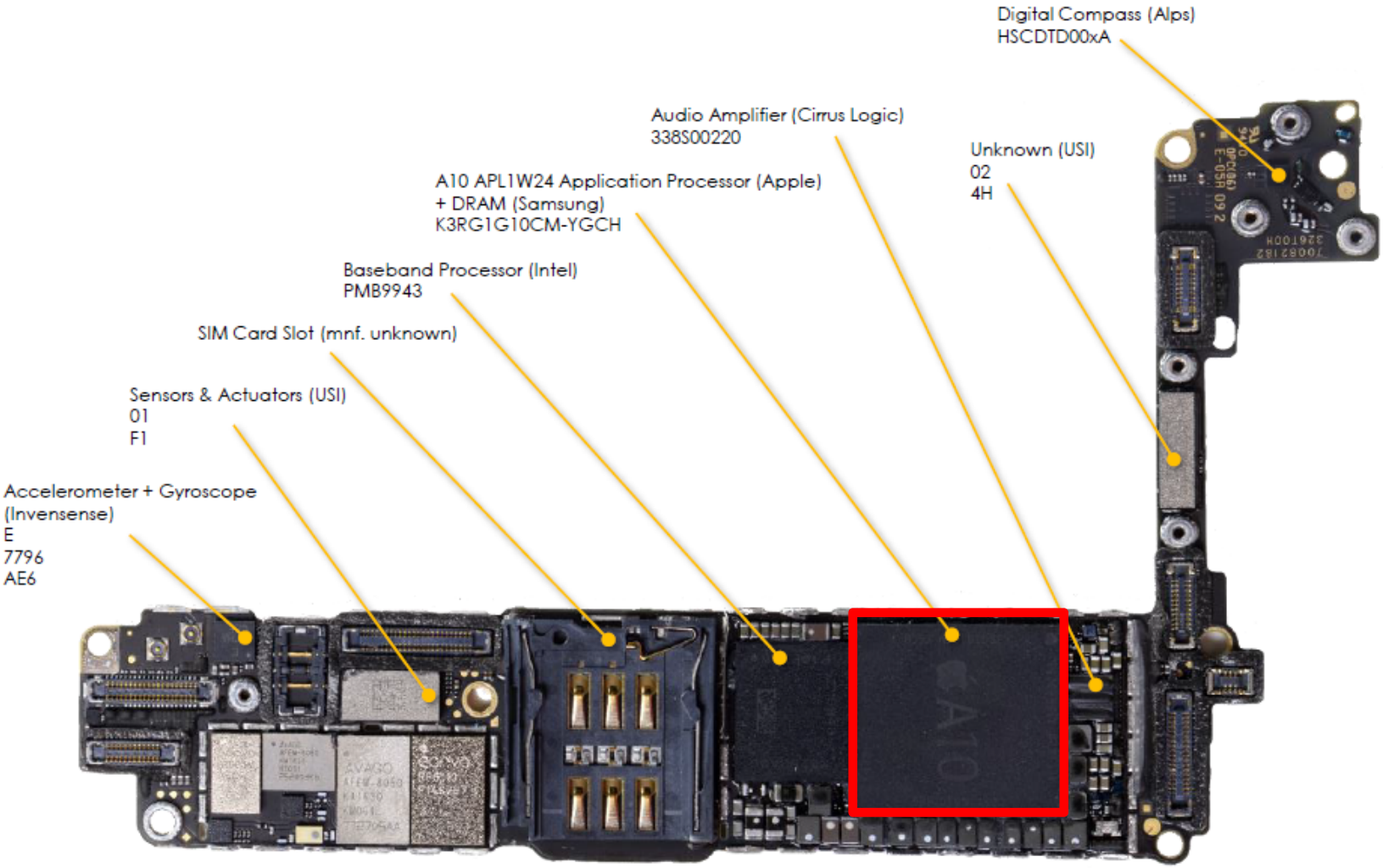


Photo from TechInsights-ChipWorks

Apple A10 Application Processor

Photo from Tech Grapple

- Two 2.34-GHz 64-bit CPUs
- Also, two low-power CPUs
- Six-core graphics processor
- TSMC 16-nm FinFET CMOS
- 125 mm^2 die area
- 3.3 billion transistors

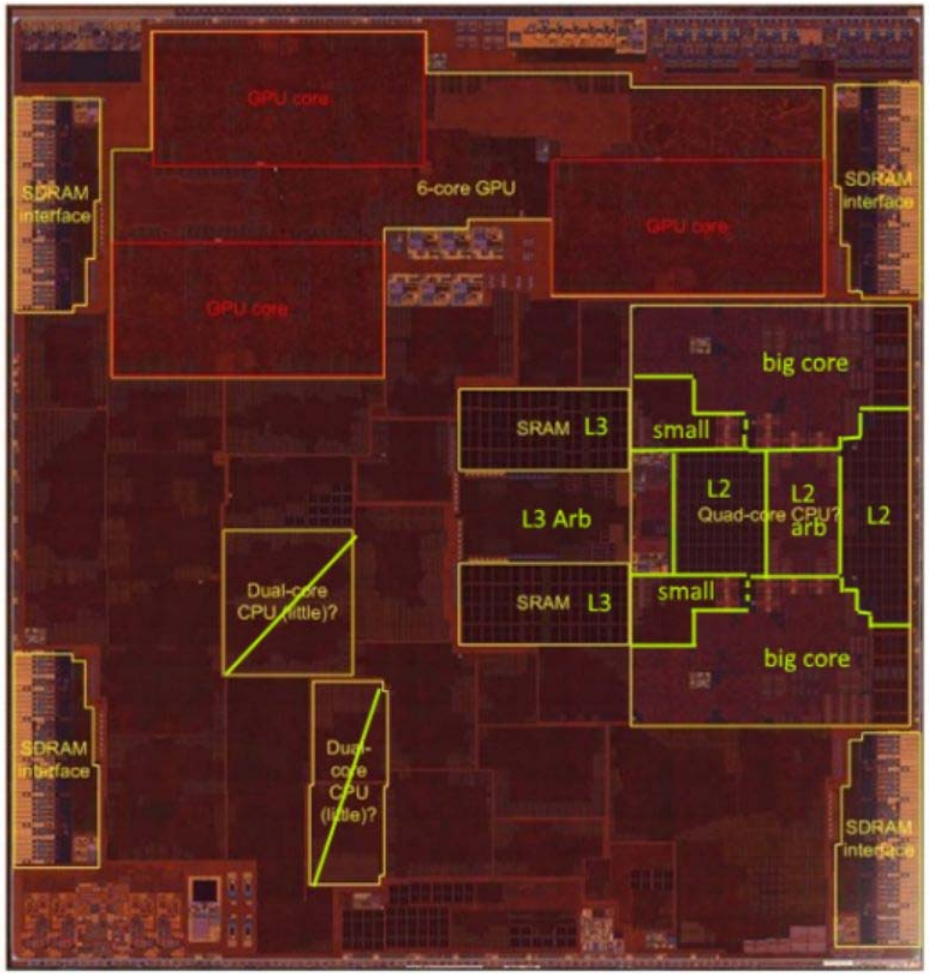
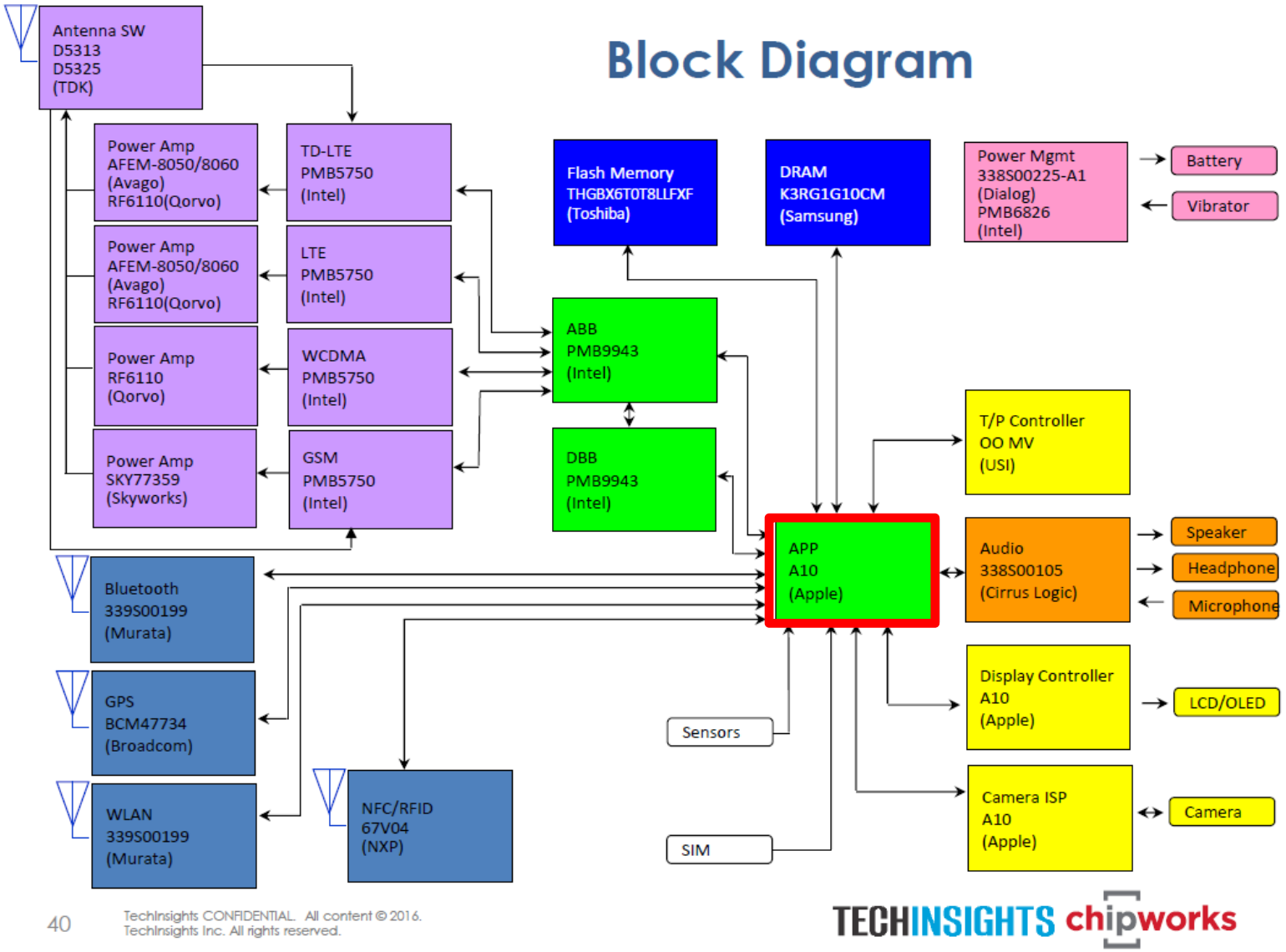


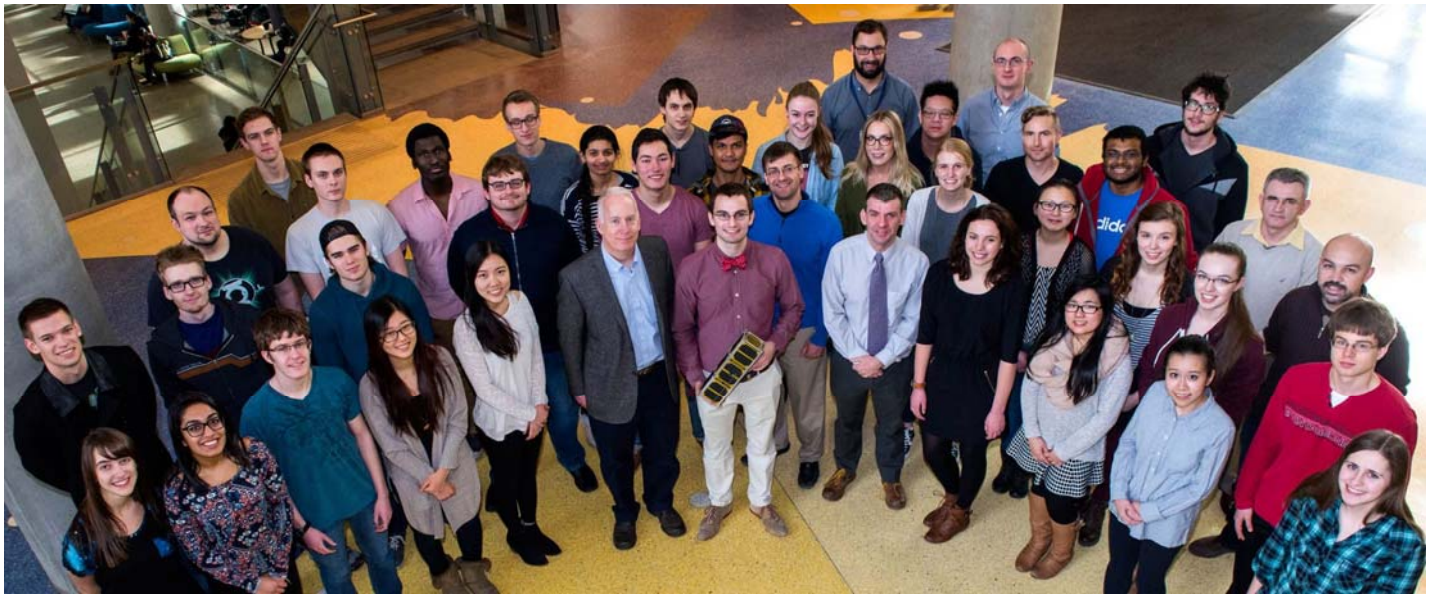
Photo from TechInsights-ChipWorks

Apple iPhone 7 System Architecture

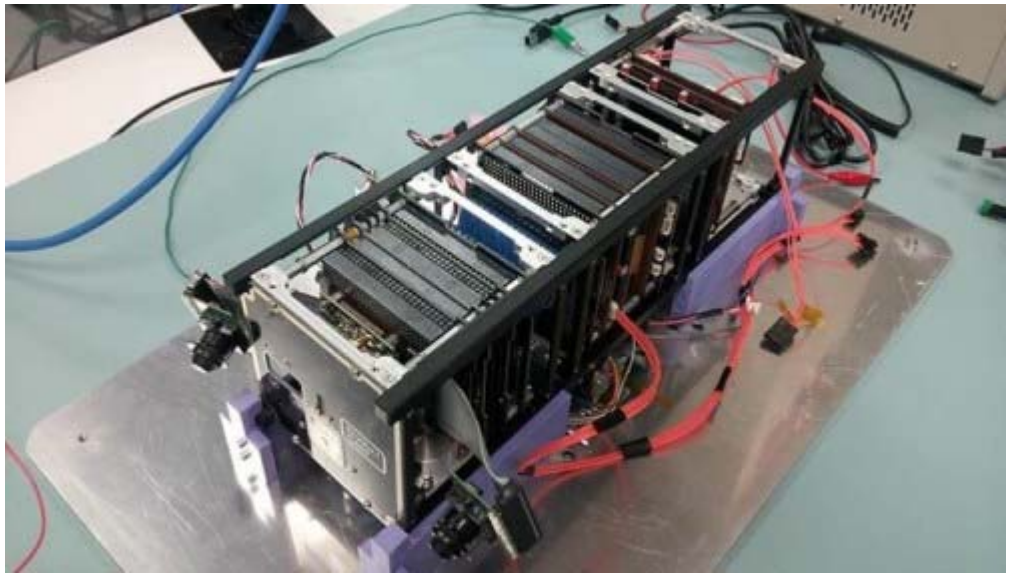


“Ex-Alta 1” CubeSat Satellite Designed at the U of A

- A 10 × 10 × 30 cm^3 cube satellite
- Launched to the Int’l Space Station May 26, 2017. Ejected into orbit May 28, 2017.
- The orbit decayed on Nov. 14, 2018
- Tested an experimental digital fluxgate magnetometer (Dept. of Physics, U of A)
- Participated in a multipoint space plasma physics experiment



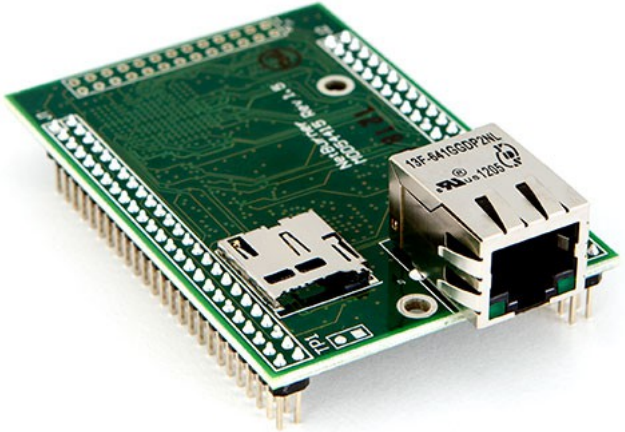
Ex-Alta 1 Embedded System Details



- The basic architecture is a 3U CubeSat platform (Innovative Solutions in Space, Delft, the Netherlands).
- Microcomputer: PC104 class (9.0 cm × 9.6 cm PCB) NanoMind A721D (GomSpace A/S, Aalborg East, Denmark) with 8 to 40-MHz 32-bit ARM7 CPU
- Memory: 2 MB RAM, 4 MB code flash, 4 MB data flash, 2 Gbyte SD card
- Comms.: Two I2C busses at 400 kbit/s using 68-byte transmit & receiver buffers
- Software environment: FreeRTOS and NanoMind software libraries

The ECE 315 Embedded System

- The NetBurner MOD5441X “Ethernet Core Module” is a modern (from 2013) 32-bit microcomputer based on the 250-MHz Freescale MCF54415 microcontroller unit (from 2011). This System-on-a-Chip (SoC) has a 250-MHz, 32-bit ColdFire V4 µP plus a large number of supporting subsystems: SDRAM controller, 10/100 Mbps Ethernet controller, 10 serial UARTs, 4 DMA-supported SPI interfaces, 8 12-bit ADCs, 2 12-bit DACs, 42 digital I/Os, etc.



NetBurner MOD54415-100IR
Ethernet Core Module

NetBurner MOD-DEV-70CR
Development Board

The NetBurner MOD54415-100IR Module

- The daughter card is the MOD54415-100IR Ethernet Core Module,
- The MOD54415-100IR is an embedded controller that has a 32-bit CPU (in the MCF54414 Microcontroller Unit), 32-Mbyte flash memory, 64-Mbyte SDRAM, and a 10/100 Mbps Ethernet interface.
- It also has numerous programmable peripheral interfaces.

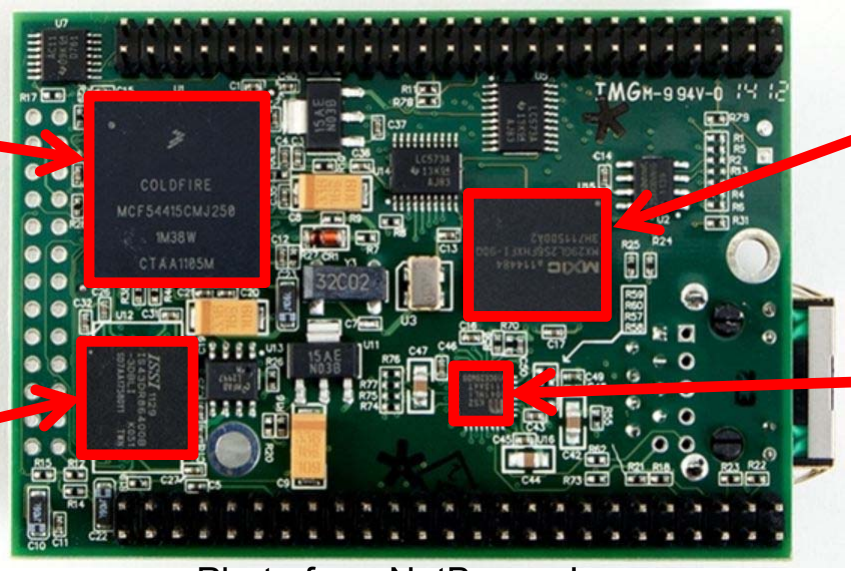


Photo from NetBurner Inc.

MCF5441X Microcontroller Architecture

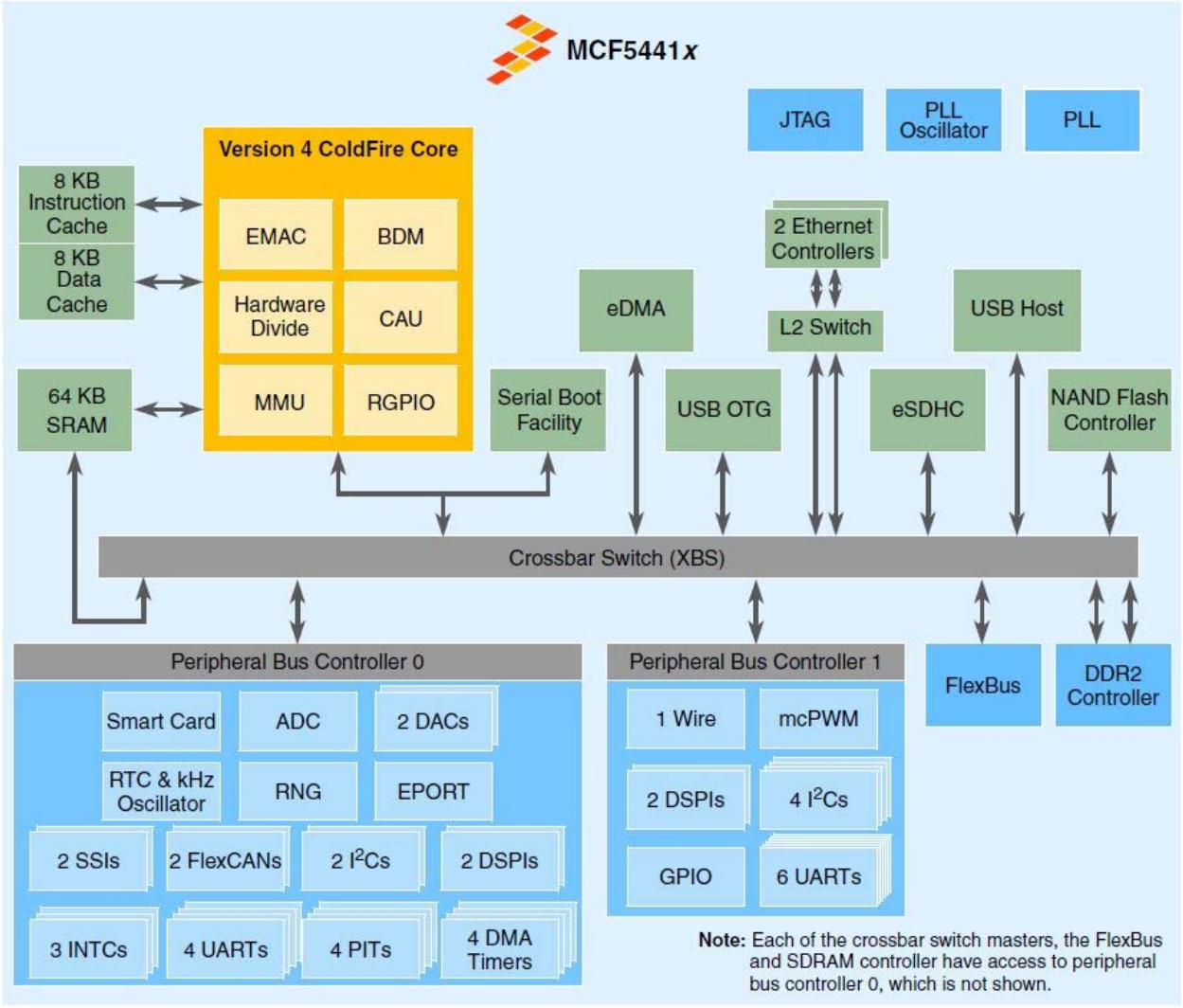


Diagram courtesy of Freescale Semiconductor Inc.

What are “Real-time Embedded Systems”?

- A ***real-time embedded system*** is an embedded system that is fast enough to react to changes in the environment so that the surrounding system that it is responsible for controlling is indeed controlled correctly, safely and efficiently.
- An embedded system may fail to operate “in real time” for several possible reasons:
  - The processor is too slow for the required workload.
  - The software was inefficiently designed, or has become overburdened over time due to upgrades.
  - The real-time requirements of the system have been increased over time to become excessively demanding.

Hard Real Time versus Soft Real Time

- It is common practice to distinguish between hard real time and soft real time embedded systems.
- A ***hard real-time embedded system*** is an embedded system where violations of real-time constraints would definitely be unacceptable. Constraint violations could cause injury or death to humans, equipment damage, loss of material, or serious financial loss.
- A ***soft real-time embedded system*** is an embedded system where occasional violations of the real-time constraints would not be desirable, but could be tolerated. For example, a vending machine or an Automated Teller Machine (ATM) can tolerate small delays with only minimal occasional frustration to human users.

Performance Measures for Embedded Systems

- ***Algorithm correctness:*** Were the intended algorithms correctly applied to the system inputs and stored data to produce the desired system state changes and outputs?
- ***Response time:*** Did the embedded system respond fast enough to input changes (from external signals & user commands) with the appropriate updated outputs?
- ***Resource efficiency:*** Was the consumption of electrical power, natural gas, water, other chemicals, communication bandwidth, etc. minimized when accomplishing the work?
- ***Initial cost:*** How much did it cost the customer to buy and/or build and then install the embedded system?

Other Important Performance Measures

- ***Predictable or deterministic response time:*** Is the response time sufficiently predictable (that is, have a small variance) as well as being acceptably fast?
- ***Well-structured design:*** Was the system architecture (both software and hardware) designed to minimize the engineering costs, to minimize the introduction of design errors, and to minimize the costs of implementing future engineering change orders (ECOs)?
- ***Total lifetime operating cost:*** How much will it cost to operate the system over its expected lifetime, including maintenance, upgrade & repair costs, training costs for personnel, recycling & disposal costs, etc.?

Safety Critical Systems

- Embedded systems are increasingly placed in control of systems that could cause unacceptable damage and loss in the event of technical failure. Examples: medical equipment, transportation systems, nuclear reactors & weapons systems.
- In *safety critical design* the system is designed so that, for all of the expected failure scenarios, the overall system will either continue to operate (perhaps with reduced functionality) or be shut down safely by the embedded system in such a manner that damage and loss to people, to equipment, and to the environment are avoided or at least minimized.
- In a *fail-safe design*, a serious failure will cause the system to be shut down and placed in a safe state.
- A *fault-tolerant system* is a system that can recover from a wide variety of failures and continue operating. Typically such a system has duplicated or redundant hardware.

Joint Human-Computer Control

- Embedded systems are increasingly used to enhance and/or to assist in the control of systems along with humans. Ex:
  - Power steering, cruise control, antilock brakes in cars
  - Fly-by-wire in aircraft; autopilot in ships and vehicles
- Special care must be paid to ensure that the assistance provided by an embedded system will never worsen an operating situation (especially an emergency situation) in its interaction with human operators.
- The embedded system should clearly warn the human operator of dangerous situations. Embedded system control should be possible for the trained human operator to disable or override. There are few situations where an embedded system should be allowed to override a human operator.

Why is Computer Interfacing important?

- Embedded systems must interact with an analog world.
  => digital-analog & analog-digital interfaces are required
- Digital systems are constructed using digital subsystems.
  Few computer engineers design hardware from scratch.
  => digital-digital interfaces between subsystems are required
- Software needs to initialize, control, & monitor the hardware.
  => software-hardware interfaces (e.g., drivers) are required
- Software systems are constructed from software subsystems.
  => software-software interfaces needed (e.g., function calls)
- Human users judge digital systems by the user interface.
  => user interface design is critical to product success

Typical Computer Interfacing Activities

1) Selecting software/hardware subsystems that can at least potentially work well with each other.

2) Providing appropriate hardware-hardware connections. Select standards, connectors, cabling, drivers, receivers, etc.

3) Configuring hardware interfaces using switches, jumpers, firmware, software, network settings, device drivers, etc.

4) Configuring software by selecting compatible software versions, invoking compilers and linkers with the correct parameters, enabling/disabling conditionally-compiled modules, calling drivers with the correct parameters, etc.

5) Designing any new “glue” hardware and software that might be required to get subsystems to interface correctly.