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 communication interfaces

- This course considers the design and debugging of systems that involve the interaction of microcomputer ***hardware***, ***embedded software***, and ***communication 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 much more than a factor of 10) by the larger number of embedded systems in our environment.
- 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 is usually absent in an embedded 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 great *consistency*, *efficiency*, and *safety*.
- A wide variety of technologies are used to implement industrial ***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

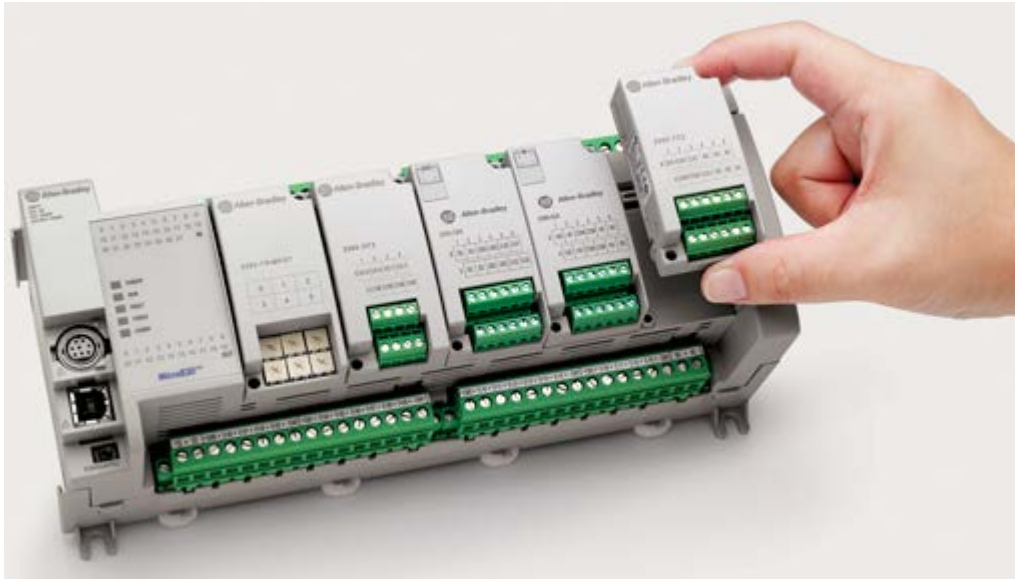


Photo courtesy of Rockwell Automation

- 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

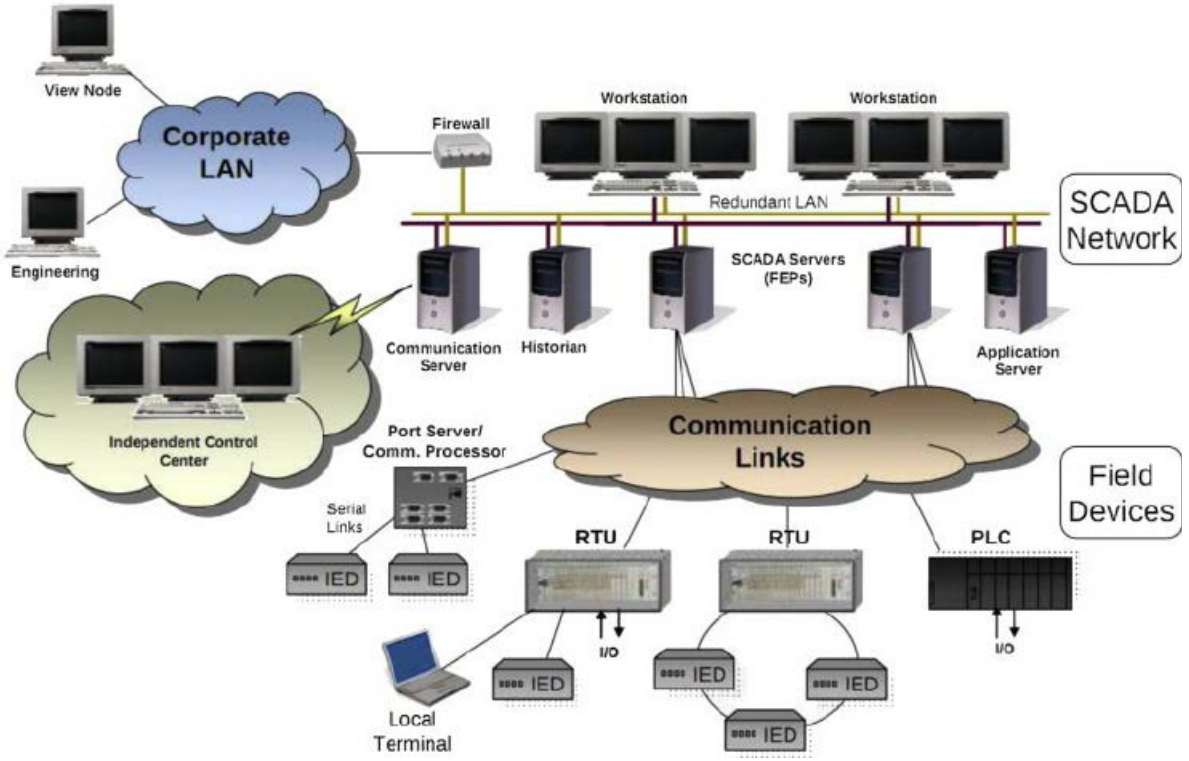


Figure courtesy of Dr. Helge Janicke, De Montfort Univ.

- ***SCADA systems*** are typically geographically distributed systems that coordinate and supervise *remote terminal units* (RTUs) and *programmable logic controllers* (PLCs) over a *communication 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 Kevin 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 ***networked embedded systems technology,*** which has been growing in importance over many years.
- The current popularity of the term Internet of Things is serving a useful purpose in that it highlights both the growing ***opportunities*** as well as the ***challenges*** and ***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, edge computing
- Also, fog computing, mist computing, etc.
- Ubiquitous computing (ubicomp), pervasive computing
- Smartphone, smart city, smart grid, smart home, etc.
- Ambient Intelligence

New terms come into fashion all the time. Marketing using new terms gains attention to new product lines, new conferences, new training courses, etc. Often the new technology is an evolution of existing tech.

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 is a state machine that *fetches*, *decodes* & *executes* software instructions.

- A ***microprocessor*** (µP) is a digital system that contains a CPU and closely related subsystems such as: *instruction pre-fetching unit*, *branch prediction unit*, *interrupt handling system*, and *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, communication interfaces) that are programmed using software to implement the desired embedded system behaviour.

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), instrumentation, factories, robots, numerical controlled tools, refineries, oil fields, gas plants, ...

Why Use Embedded Systems?

- Embedded systems provide high-speed, ***software-program-mable*** sensor monitoring and signal processing, data processing, actuator control, communication 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 promote ***new applications of embedded systems*** that grow their revenues.

The Jacquard Machine (1804)

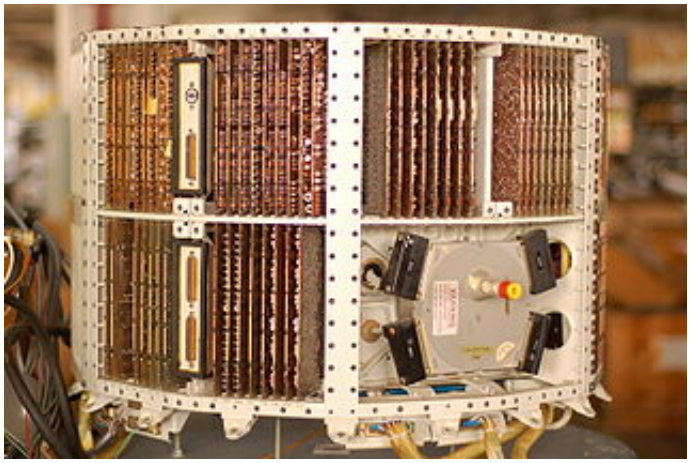
- A **loom** is a complex mechanical machine that is used to weave cloth and tapestry.
- For centuries there has been commercial demand for woven products with intricate patterns. Creating these patterns requires complex mechanical movements that are challenging and tedious to produce under the manual control of human weavers.
- The **Jacquard machine**, invented by Joseph Marie Jacquard in 1804, is a **punch-card programmable mechanical embedded system** that controls a loom so that it automatically weaves textiles with complex and often repeating patterns.
- Jacquard's invention built upon simpler loom controllers created by Bouchon (1725), Falcon (1728) and Vaucanson (1745), also from France.

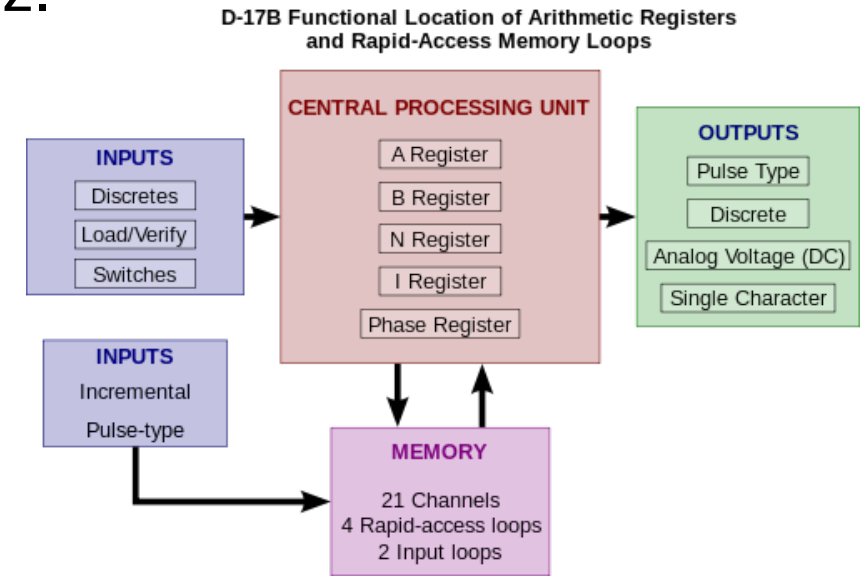
Image courtesy of Wikipedia

Loom controlled by a Jacquard Machine

Early Electronic 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.

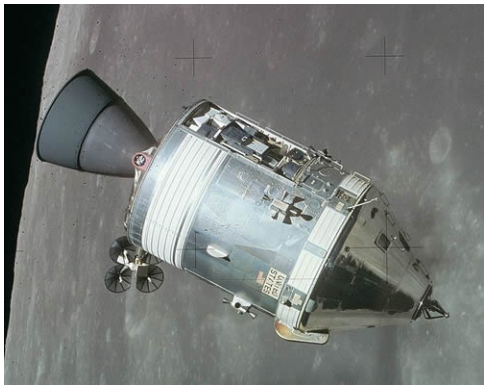




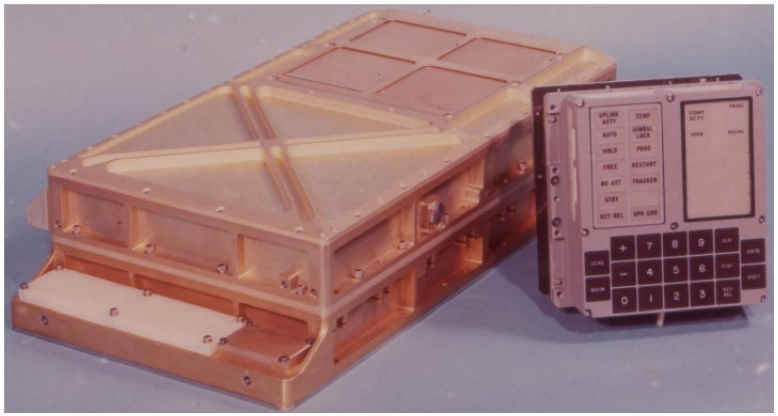
Data and images from www.wikipedia.org

Early Electronic Embedded Systems (2)

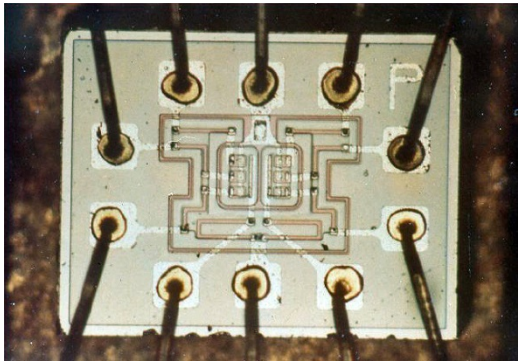
- 1966: MIT Instrumentation Laboratory / Raytheon Apollo Guidance Computer (AGC), used to control the Apollo spacecraft. Weighing 32 kg, the AGC contained 2800 integrated circuits (dual 3-input NOR gates) and dissipated 55 W with a 4-phase 1.024-MHz clock.



Apollo Command & Service Modules



AGC Computer and DSKY User Interface



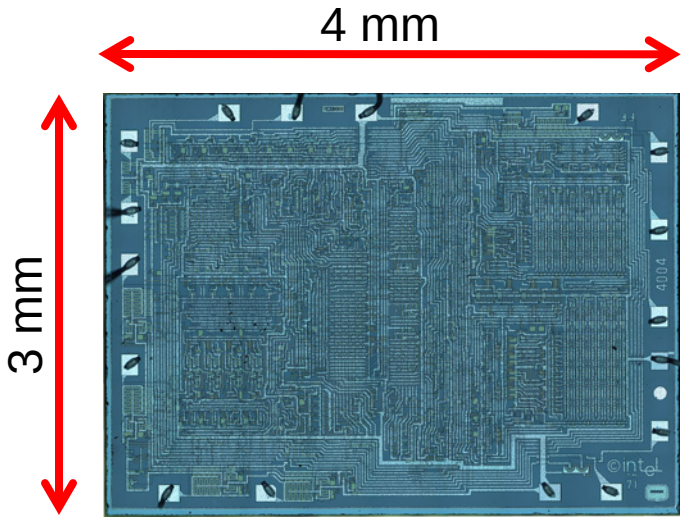
RTL Dual 3-input NOR IC

Data and images from www.wikipedia.org

Early Electronic 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

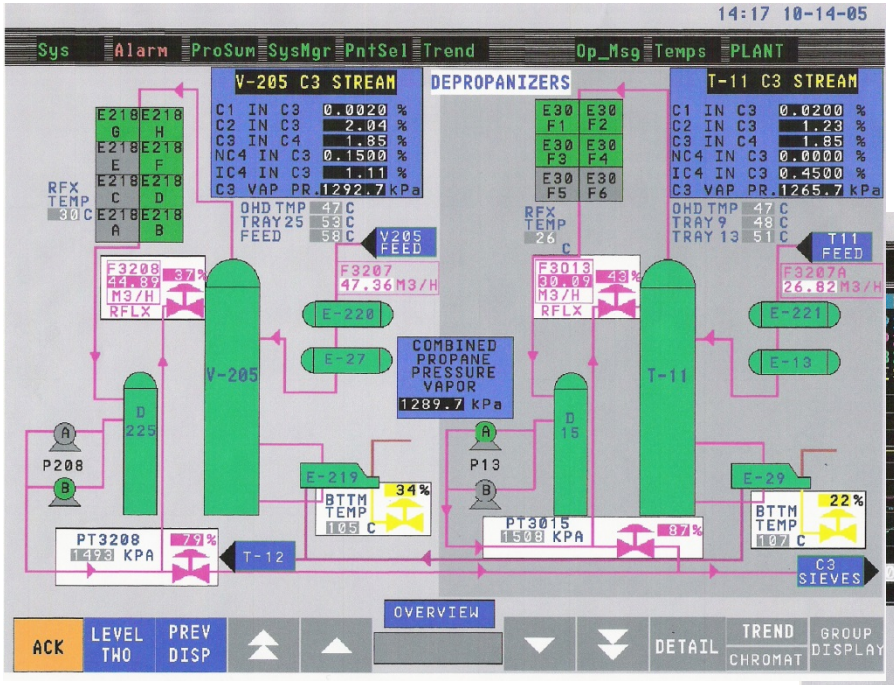
- Canada's largest oil refinery, capable of processing 320,000 barrels of oil per day
- Computer control is required to maximize production efficiency and ensure safety.

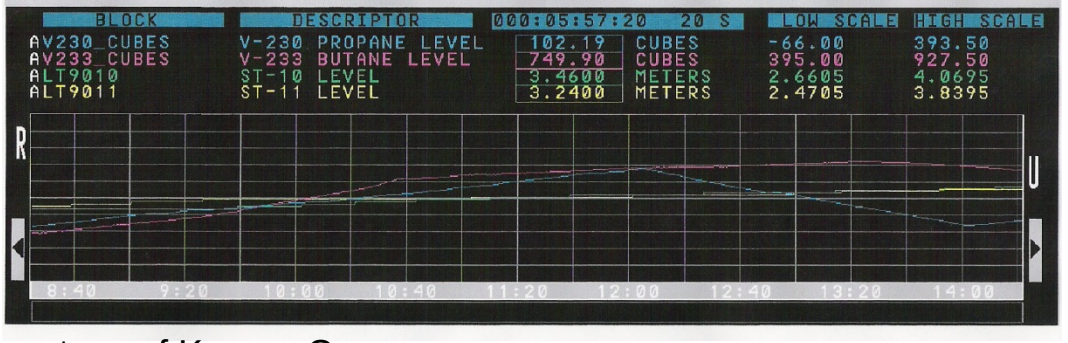
Photos from the Globe and Mail

Keyera's Gas Complex in Rimbey, AB

Photo courtesy of Keyera Corp.

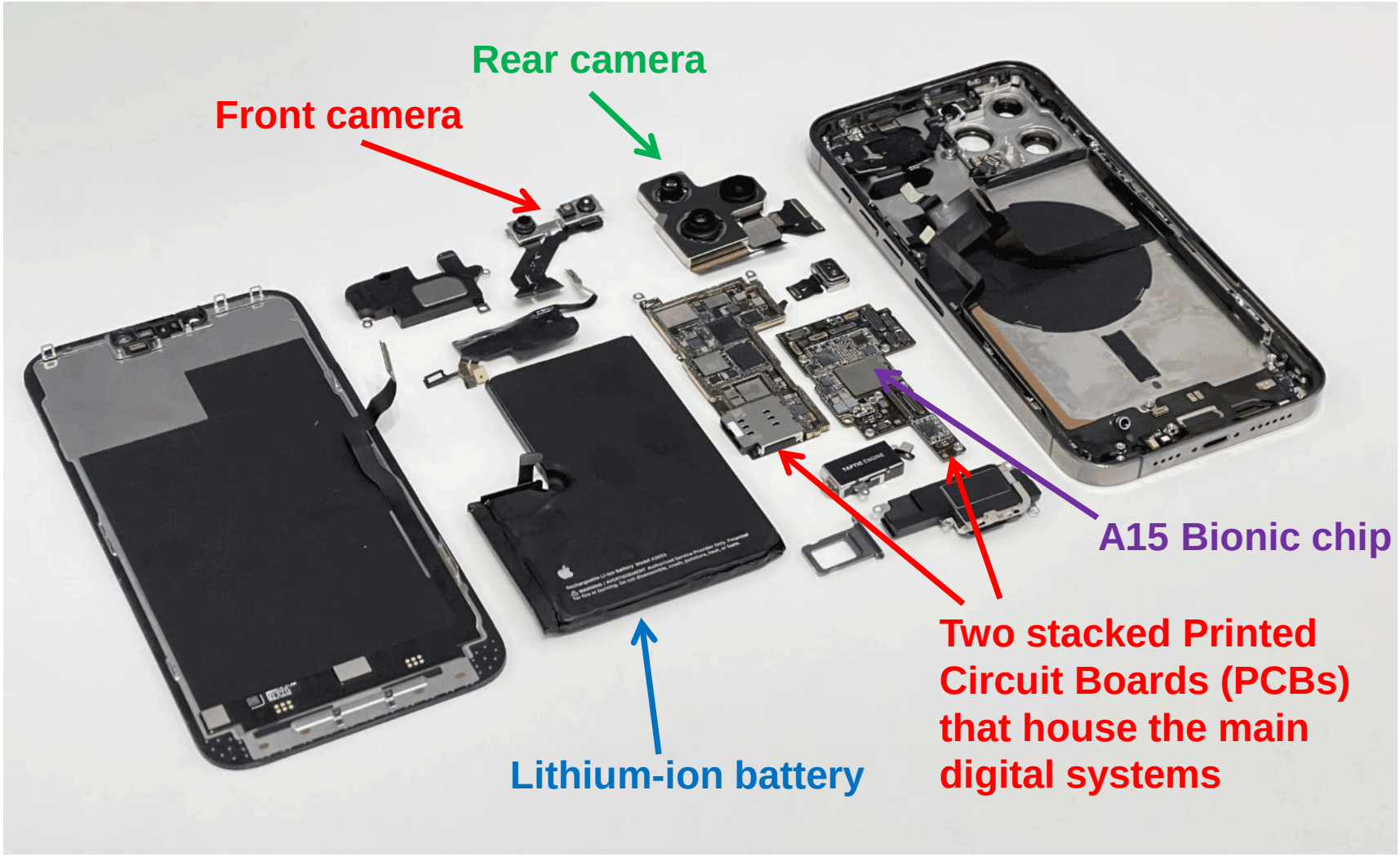
Control Console Windows at the Gas Plant





Photos courtesy of Keyera Corp.

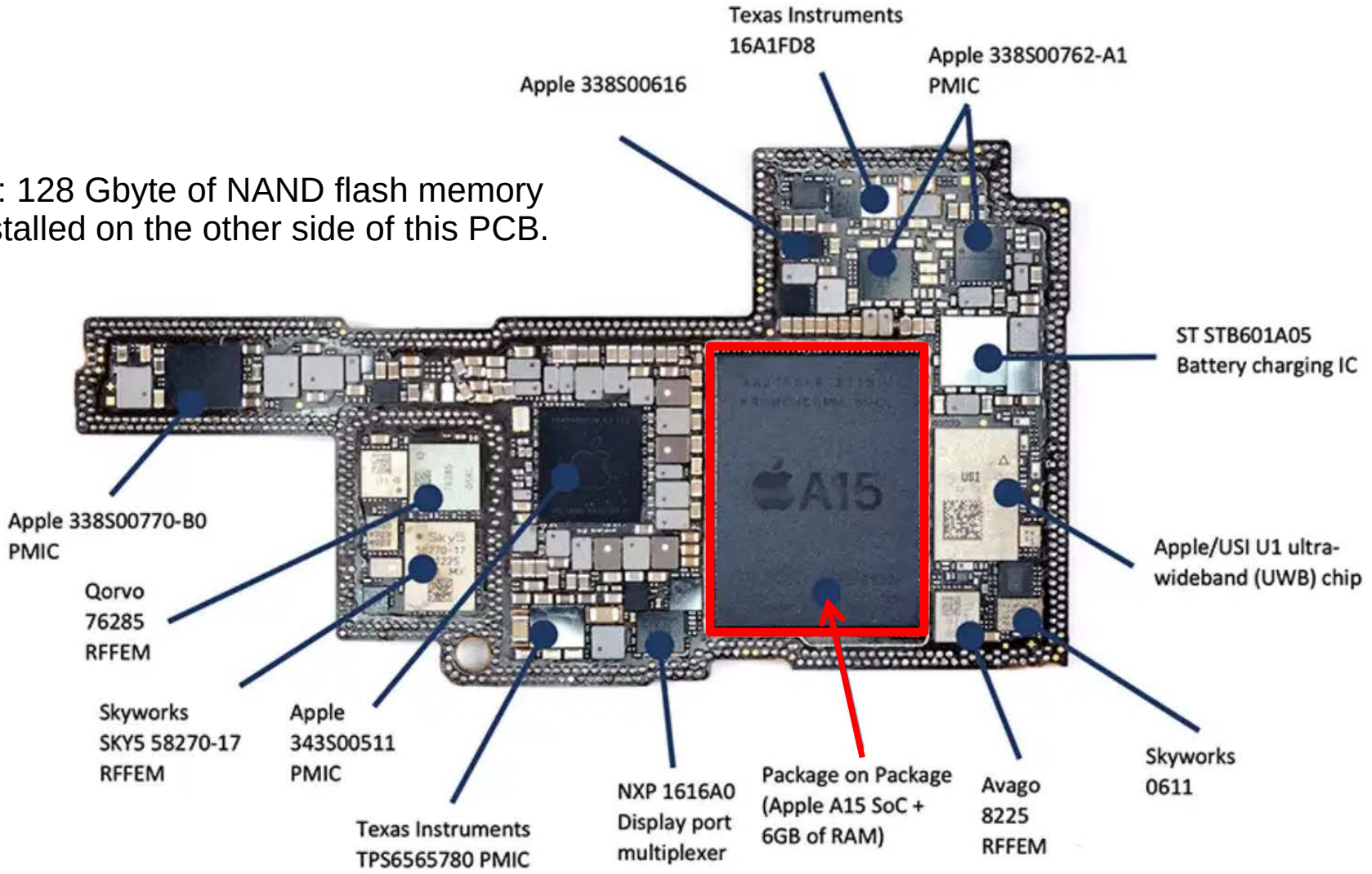
Apple iPhone 13 (2021) Teardown



Photos from UnitedLex

Apple iPhone 13: Top side of one of two PCBs

Note: 128 Gbyte of NAND flash memory is installed on the other side of this PCB.



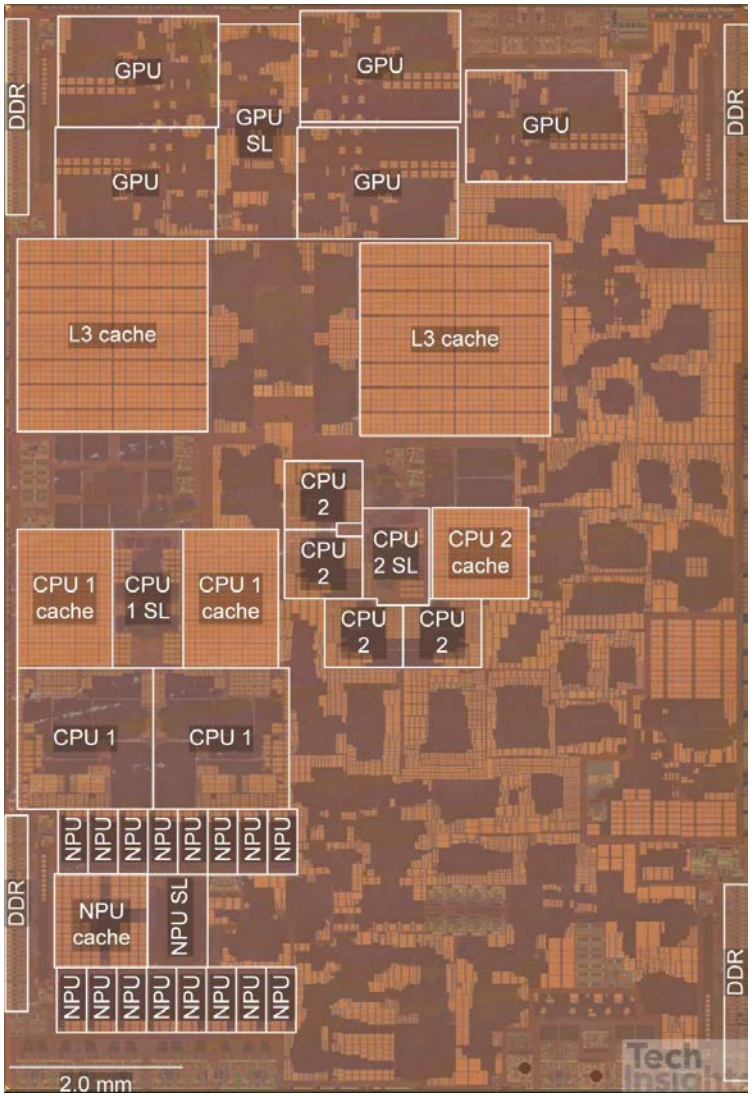
***Note*: SoC = "System on a Chip"**

Photo from UnitedLex

Apple A15 Application Processor

Photo from Wikipedia

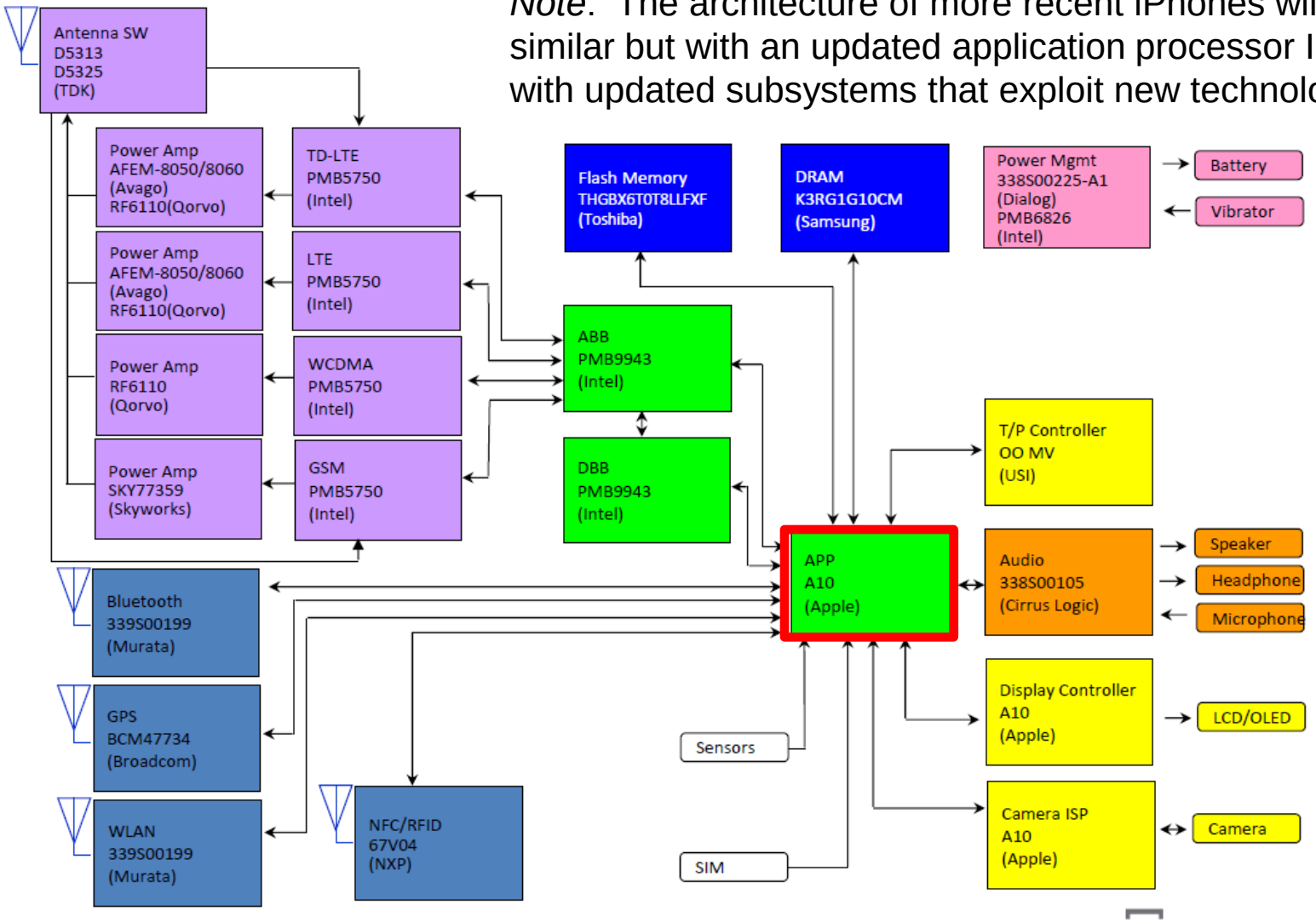
- 107.7 mm^2 silicon area in the main SoC die
- TMSC N5P 5-nm FinFET CMOS process
- ~15.8 billion transistors
- Two 3.23-GHz 64-bit ARMv8-class CPUs
- Also four low-power 32-bit CPUs
- Five graphics processor unit (GPU) cores
- 16 neural processor unit (NPU) cores
- Four DDR4 SDRAM dies (6 GBytes) are stacked over the application processor die.



Die photo from TechInsights

Apple iPhone 7 System Architecture (2016)

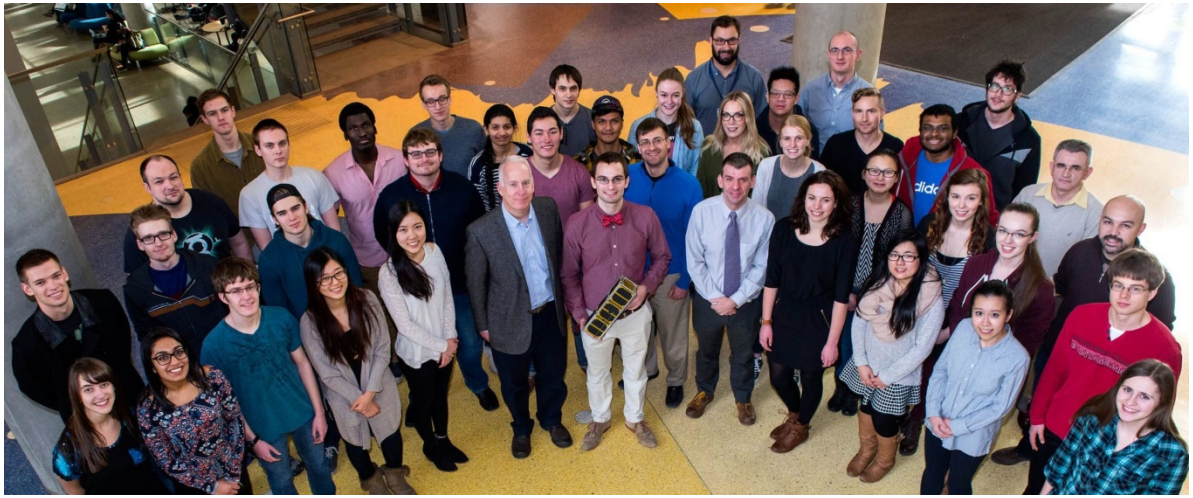
Note: The architecture of more recent iPhones will be similar but with an updated application processor IC and with updated subsystems that exploit new technology.



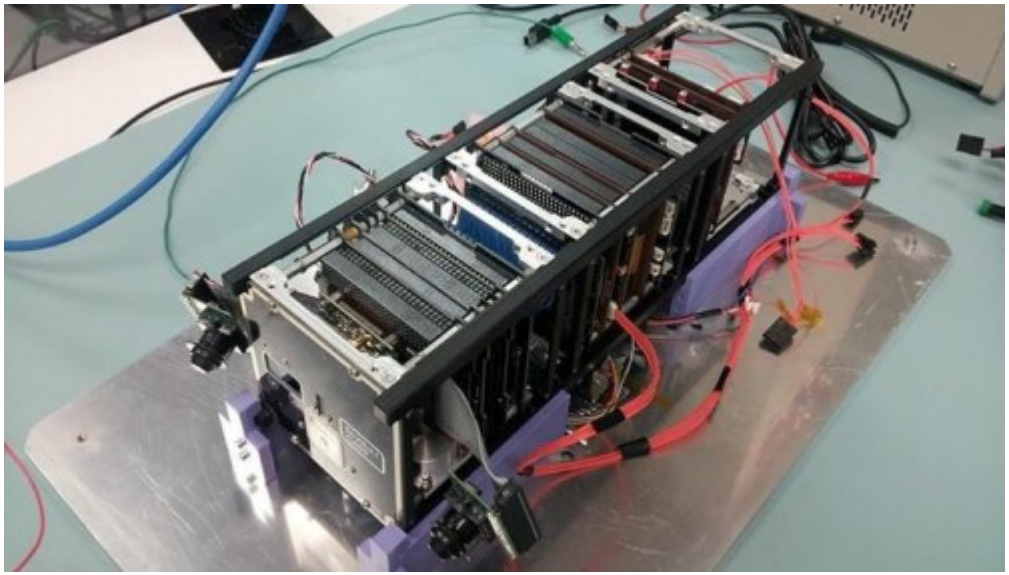
TECHINSIGHTS chipworks

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

- A 10 × 10 × 30 cm^3 cube satellite, that is, 3U
- 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 dual fluxgate magnetometer (Dept. of Physics, U of A)
- Participated in a multipoint space plasma physics experiment



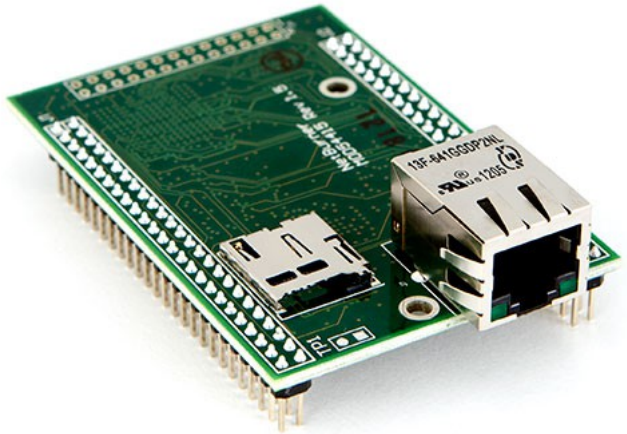
Ex-Alta 1 Embedded System Details



- 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 I^2C busses at 400 kbit/s using 68-byte transmit & receiver buffers
- Software environment: FreeRTOS and NanoMind software libraries

The Old 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.

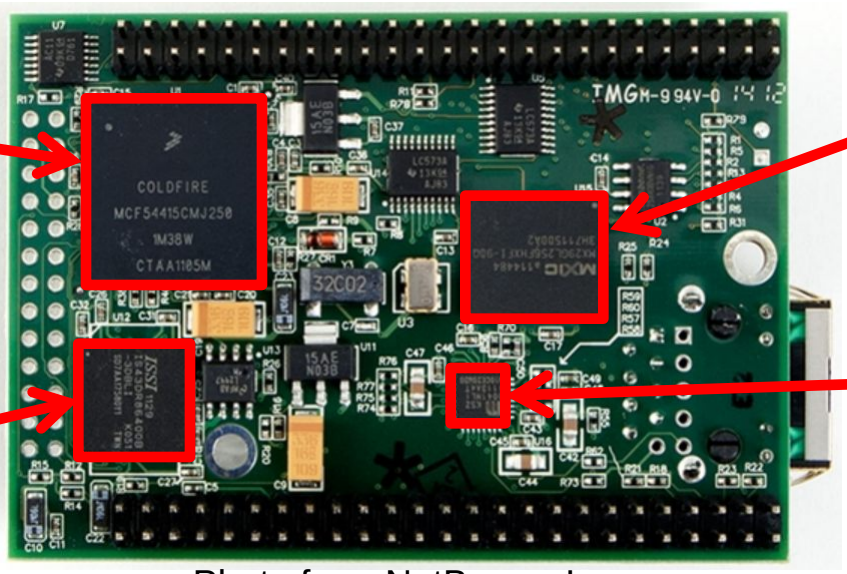


Photo from NetBurner Inc.

MCF5441X Microcontroller Architecture

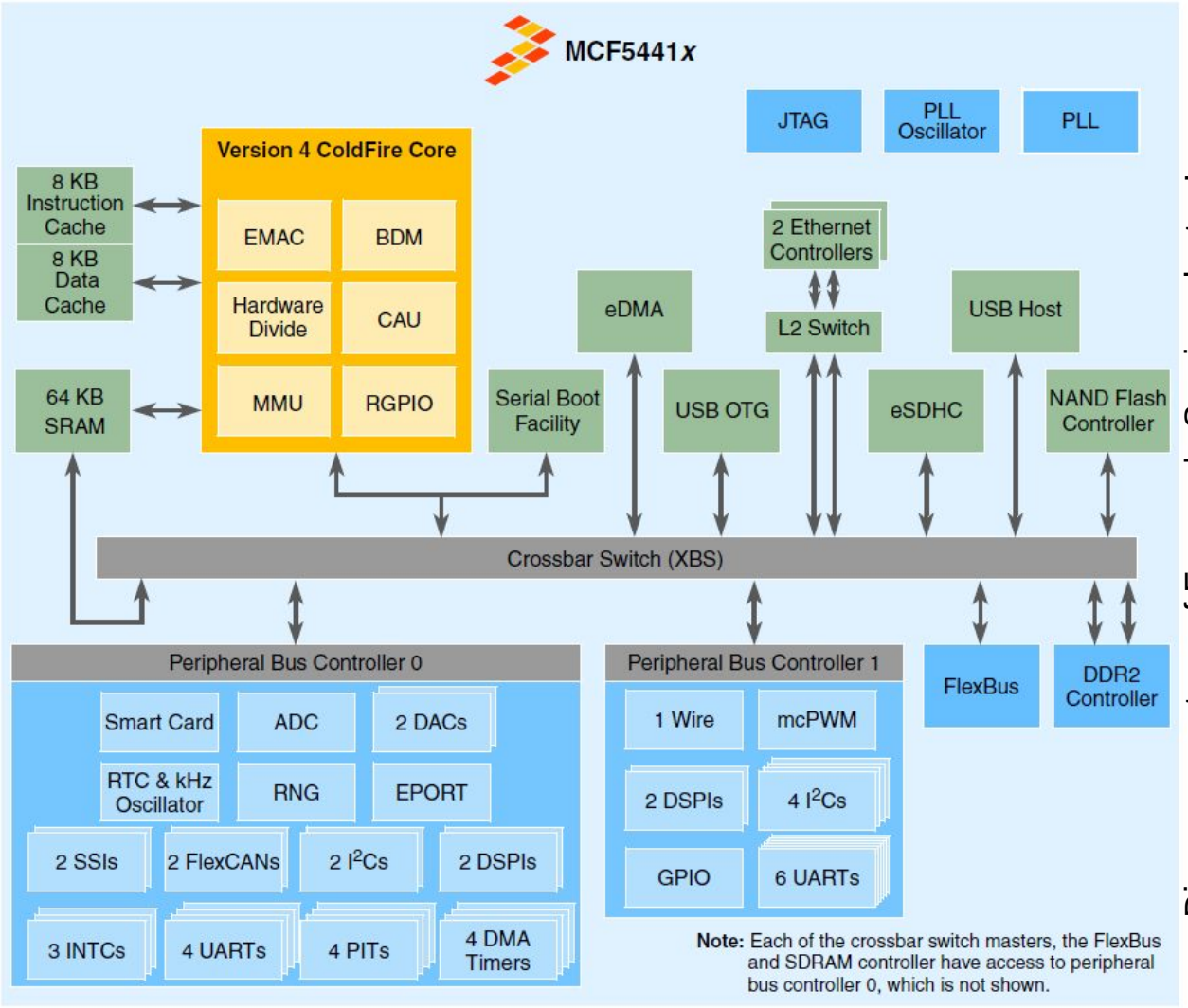
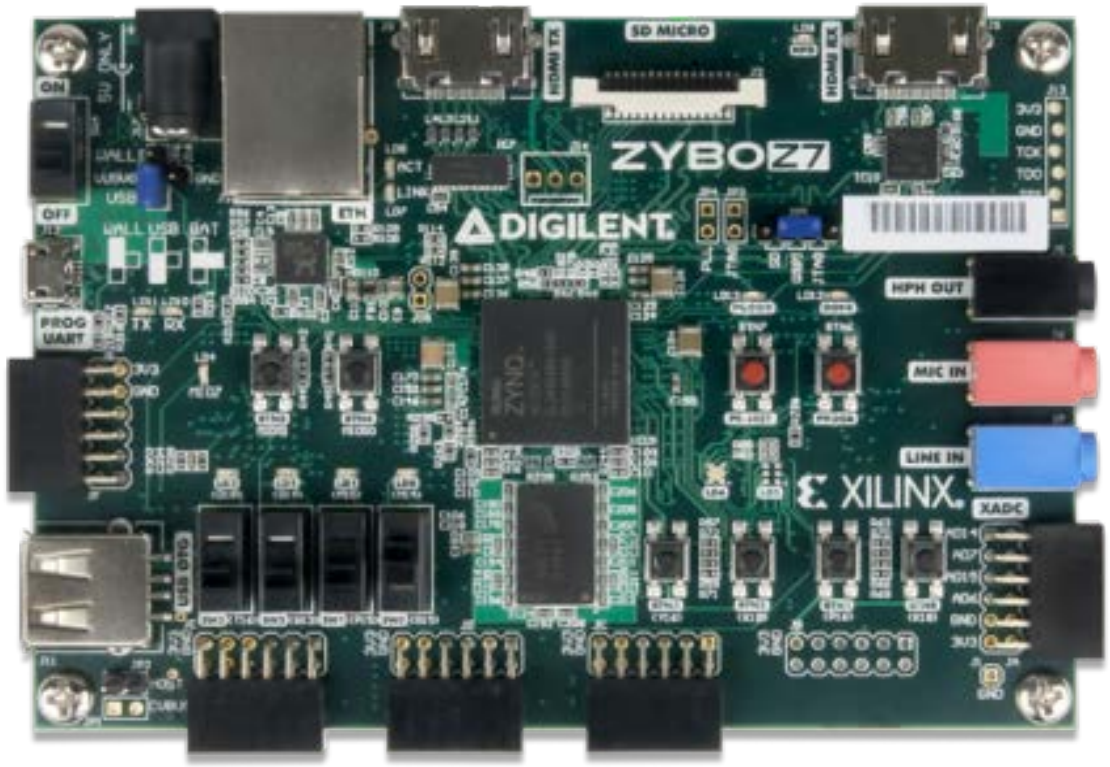


Diagram courtesy of Freescale Semiconductor Inc.

The New ECE 315 Microcontroller System



Courtesy of the Digilent Inc.

This is the same Zybo Z7-10 FPGA board as used in the ECE 410 lab. It contains the **Zynq Z-7010 FPGA**, which has 17.6K LUTs, 35.2 Kffs, 2.1 Mb of RAM, 80 DSP slices, two 12-bit DACs & **two 32-bit ARM A9 CPUs**.

Xilinx Zynq-7000 Family SoC FPGAs (2014)

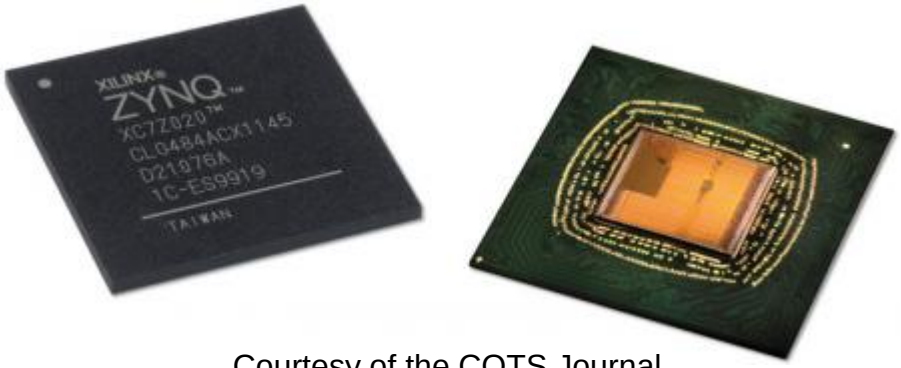
ECE 315

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial	-1			-1			-1			-1
	Speed Grades: Extended	-2			-2,-3			-2,-3			-2
	Speed Grades: Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

Xilinx Inc., "Zynq-7000 SoC Product Selection Guide," Doc. XMP097, 2014-19.

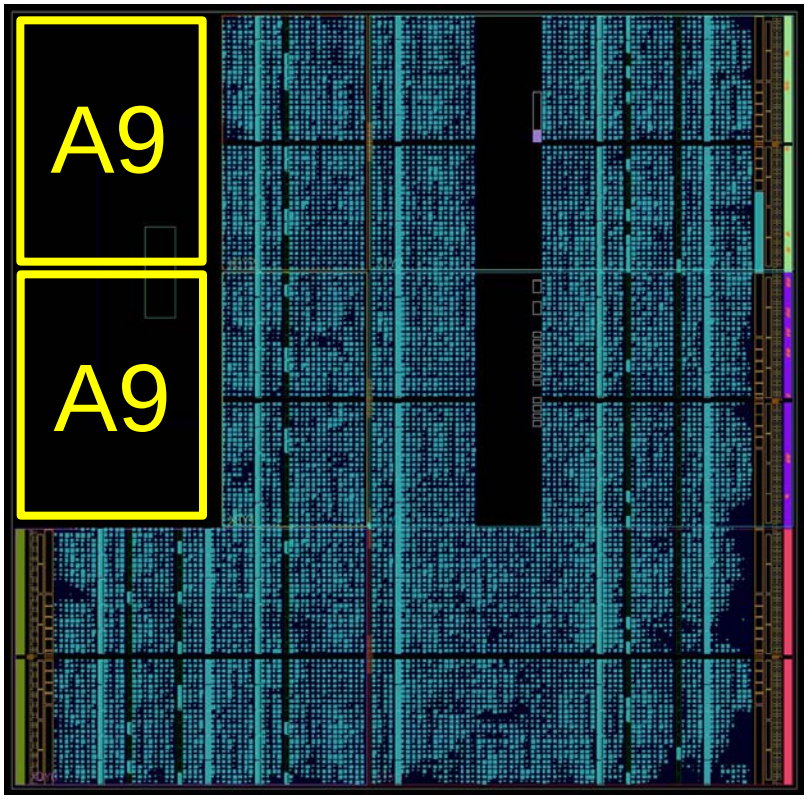
Xilinx Zynq Z-7020 SoC FPGA (2014)

Packaged FPGA Device



Courtesy of the COTS Journal

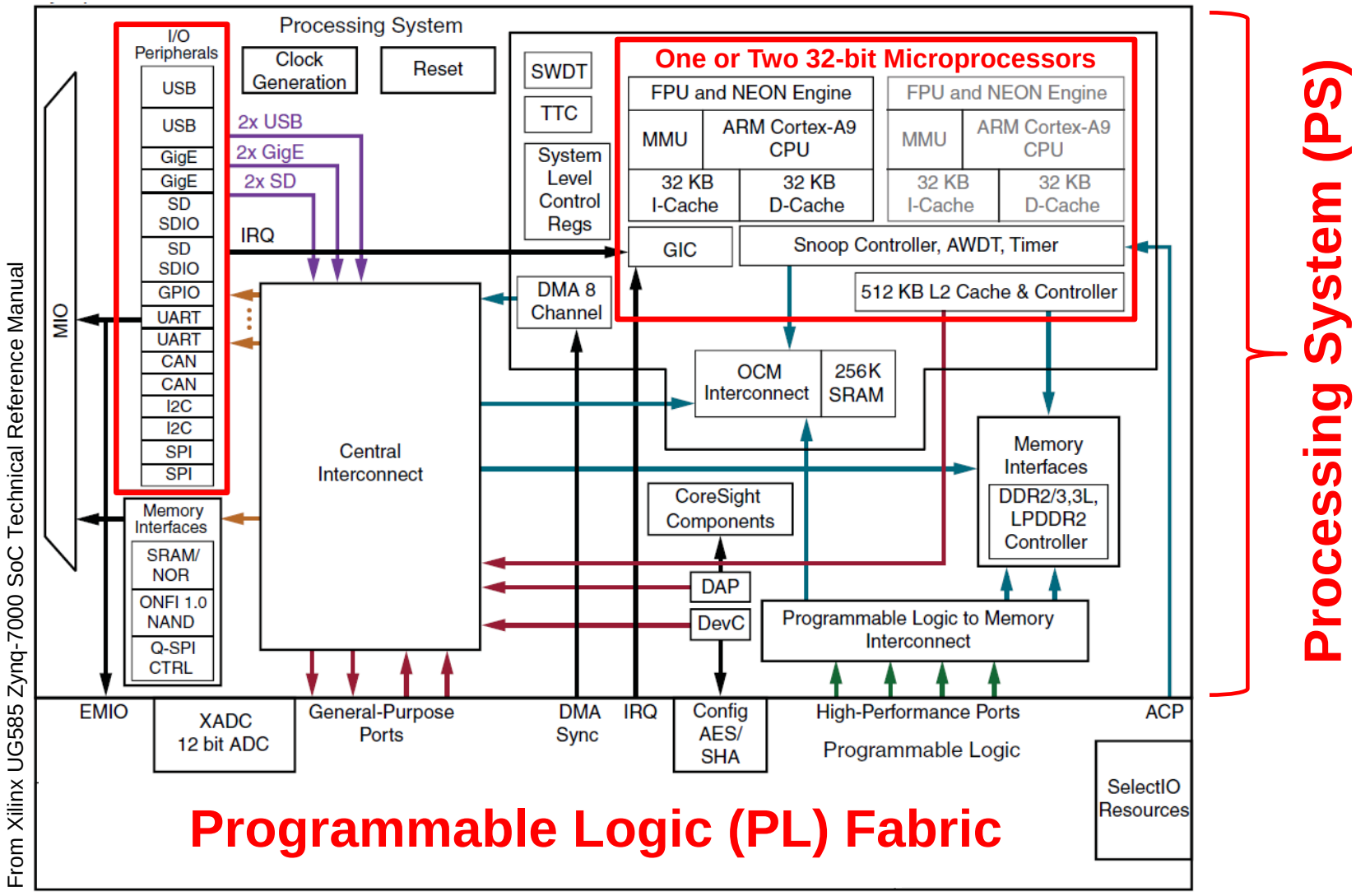
CAD Layout View



Features of the Zynq Z-7020

- 53,200 LUTs
- 106,400 flip-flops
- 4.9 Mb of Block RAM
- 220 DSP slices
- PCIe interface, Gen2 x4
- Two 12-bit D/A converters
- Two 32-bit ARM A9 microprocessors

Architecture of the Xilinx Zynq-7000 SoC Family



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*, *efficiently* and *safely*.
- 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?
- ***Predictable or deterministic response time:*** Is the response time sufficiently predictable (that is, have a small variance) as well as being acceptably fast?
- ***Resource efficiency:*** Was the consumption of electrical power, natural gas, water, other chemicals, communication bandwidth, etc. minimized when accomplishing the work?

Other Important Performance Measures

- ***Initial cost:*** How much did it cost the customer to buy and/or build and then install the embedded system?
- ***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.? Will the vendor of the embedded system still be around in 10 years?
- ***Well-structured design:*** Was the system architecture (both software and hardware) designed so as to minimize the engineering costs, to minimize the introduction of design errors, and to minimize the costs of implementing future engineering change orders (ECOs)?
- ***Compliance with industry standards:*** e.g., quality, safety

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 *placed in a safe state* and *shut down*.
- A ***fault-tolerant system*** is a system that can recover from a wide variety of failures and then continue to *operate normally*. 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
- The assistance provided by an embedded system ***should 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***. Any alarms must be actively turned off.
- Control by an embedded system should be ***straightforward 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 (e.g., the operator is incapacitated).

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.