DANIELE PERI

daniele.peri@unipa.it

EMBEDDED SYSTEMS

OUTLINE

- Main topics
  - Architectures of Embedded Systems
  - Modeling, analysis, and verification of embedded software
    - Machine language and assembly language (ARM)
    - Forth

ASSESSMENT

- Each student is requested to develop a working and documented embedded application as a final project
- The project is thoroughly discussed during the oral exam
- The oral exam also targets all the topics of the course

TEXTBOOKS

- Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition, http://LeeSeshia.org, ISBN 978-1-312-42740-2, 2015.
- S. L. Harris, D. M. Harris, Digital Design and Computer Architecture: ARM Edition, Morgan Kaufmann
- L. Brodie, “Starting Forth”, http://www.forth.com/starting-forth/
- A. S. Tanenbaum, T. Austin, “Structured computer organization. 6th ed.”, Pearson
- D. A. Patterson, J. L. Hennessy, “Computer Organization and Design”, Morgan Kaufmann
- L. Brodie, “Thinking Forth”, http://thinking-forth.sourceforge.net

EMBEDDED SYSTEMS

- The most visible use of computers and software is processing information for human consumption. We use them to write books, search for information on the web, communicate via email, and keep track of financial data.
- The **vast majority** of computers in use are much less visible. They:
  - run the engine, brakes, seatbelts, airbag, and audio system in your car
  - digitally encode your voice and construct a radio signal to send it from your cell phone to a base station
  - control your microwave oven, refrigerator, and dishwasher
  - run printers ranging from desktop inkjet printers to large industrial high-volume printers
  - command robots on a factory floor, power generation in a power plant, processes in a chemical plant, and traffic lights in a city
  - search for microbes in biological samples, construct images of the inside of a human body, and measure vital signs
  - process radio signals from space looking for supernovae and for extraterrestrial intelligence
  - bring toys to life, enabling them to react to human touch and to sounds
  - control aircraft and trains
  - perform many other "invisible" tasks

These less visible computers are called **embedded systems**, and the software they run is called **embedded software**.

EMBEDDED SYSTEMS

- Although embedded systems have been in use since the 1970s, for most of their history they were seen simply as small computers
- The principal engineering problem was understood to be one of coping with **limited resources** (limited processing power, limited energy sources, small memories, etc.)
  - The engineering challenge was to optimize the designs. Since all designs benefit from optimization, the discipline was not distinct from anything else in computer science. It just had to be more aggressive about applying the same optimization techniques
- Recently, the community has come to understand that the principal challenges in embedded systems stem from their interaction with physical processes, and not from their limited resources
- The term **cyber-physical systems (CPS)** was coined to refer to the integration of computation with physical processes

EMBEDDED SYSTEMS

- In CPS, embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa.
- The design of such systems, therefore, requires understanding the joint dynamics of computers, software, networks, and physical processes. It is this study of joint dynamics that sets this discipline apart.
- When studying CPS, certain key problems emerge that are rare in so-called general-purpose computing.
  - For example, in general-purpose software, the time it takes to perform a task is an issue of **performance**, not **correctness**.
    - It is not incorrect to take longer to perform a task. It is merely less convenient and therefore less valuable.
  - In CPS, the time it takes to perform a task may be **critical** to correct functioning of the system.

EMBEDDED SYSTEMS

- Physical processes are compositions of **many things going on at once**, unlike software processes, which are deeply rooted in **sequential steps (procedural)**
- In the **physical world**, by contrast, processes are **rarely procedural**
- Physical processes are compositions of **many parallel processes**
- **Measuring** and **controlling** the **dynamics** of these processes by orchestrating **actions** that influence the **processes** are the **main tasks** of **embedded systems**
- Consequently, **concurrency** is intrinsic in CPS
- Many of the **technical challenges** in designing and analyzing **embedded software** stem from the need to bridge an **inherently sequential semantics** with an intrinsically **concurrent physical world**

EMBEDDED SYSTEMS

- The mechanisms by which **software** interacts with the **physical world** are **changing** rapidly
- Today, the trend is towards "**smart**" sensors and actuators, which carry **microprocessors**, **network interfaces**, and **software** that enables **remote access** to the **sensor data** and **remote activation** of the **actuator**
- Called variously the **Internet of Things (IoT)**, **Industry 4.0**, the **Industrial Internet**, **Machine-to-Machine (M2M)**, the **Internet of Everything**, the **Smarter Planet**, **TSensors (Trillion Sensors)**, or **The Fog** (like The Cloud, but closer to the ground), the vision is of a technology that deeply **connects** our **physical world** with our **information world**
- In the **IoT** world, the interfaces between these worlds are **inspired by** and **derived** from **information technology**, particularly **web technology**.
- **IoT interfaces** are convenient, but **not yet suitable** for **tight interactions** between the two worlds, particularly for **real-time control** and **safety-critical systems**
- **Tight interactions** still **require** technically intricate, **low-level design**
- **Embedded software designers** are forced to **struggle** with **interrupt controllers**, **memory architectures**, **assembly-level programming** (to exploit specialized instructions or to precisely control timing), **device driver design**, **network interfaces**, and **scheduling** strategies, rather than focusing on specifying desired behavior

EMBEDDED PROCESSORS

- In general-purpose computing, the variety of instruction set architectures today is limited, with the Intel x86 architecture overwhelmingly dominating all
- There is no such dominance in embedded computing

EMBEDDED PROCESSORS

- When deployed in a product, embedded processors typically have a dedicated function
- They are not asked to perform arbitrary functions with user-defined software
- Consequently, the processors can be more specialized.
  - Benefits:
    - they may consume far less energy, and consequently be usable with small batteries for long periods of time
    - they may include specialized hardware to perform operations that would be costly to perform on general-purpose hardware, such as image analysis

EMBEDDED PROCESSORS

- When evaluating processors, it is important to understand the **difference** between:
  - **Instruction set architecture** (ISA)
    - the instructions that the processor can execute and certain structural constraints (such as word size)
      - x86 is an ISA
  - A **processor realization or a chip**
    - A piece of silicon sold by a semiconductor vendor
      - There are many realizations of the x86 ISA
- An **ISA** is an **abstraction shared** by many realizations
- A single **ISA** may appear in **many different chips**, often made by **different manufacturers**, and often having **widely varying performance** profiles

EMBEDDED PROCESSORS

- The **advantage** of sharing an ISA in a family of processors is that **software tools**, which are costly to develop, may be **shared**, and (sometimes) the **same programs** may **run** correctly on **multiple realizations**
- This latter property, however, is rather treacherous, since an ISA **does not** normally **include** any **constraints** on **timing**
- Hence, although a program may **execute** logically the **same way** on **multiple chips**, the system **behavior** may be radically **different** when the processor is **embedded** in a **cyber-physical system**

EMBEDDED PROCESSORS

- Types of Processors
  - Due the **huge variety** of embedded **applications**, there is a **huge variety** of **processors** that are used.
  - They range from very **small**, **slow**, **inexpensive**, **low-power** devices, to **high-performance**, **special-purpose** devices.

EMBEDDED PROCESSORS

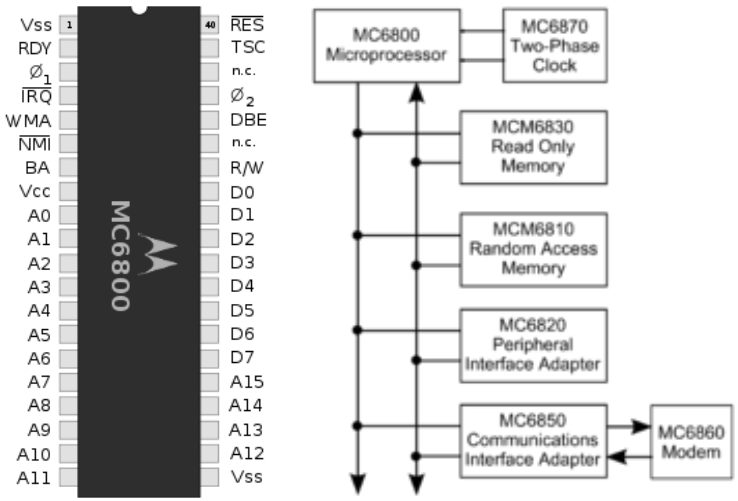
- **Microcontrollers**
  - A microcontroller (μC) is a small computer on a single integrated circuit consisting of a relatively simple central processing unit (CPU) combined with peripheral devices such as memories, I/O devices, and timers
  - **More than half** of all **CPUs sold** worldwide are **microcontrollers**, although such a claim is hard to substantiate
  - The **simplest** microcontrollers operate on **8-bit words** and are suitable for applications that require **small amounts of memory** and **simple logical functions** (vs. performance-intensive arithmetic functions)
  - They may consume **extremely small amounts of energy**, and often include a **sleep mode** that reduces the power consumption to **nanowatts**
  - **Embedded components** such as **sensor network nodes** and **surveillance devices** have been demonstrated that can operate on a **small battery** for **several years**

EMBEDDED PROCESSORS

- **Microcontrollers** can get quite **elaborate**
  - The Intel Atom, for example, is a family of x86 CPUs used mainly in netbooks and other small mobile computers.
  - These processors are designed to **use relatively little energy without losing** too much **performance** relative to processors used in higher-end computers
  - They are suitable for some embedded applications and in servers where cooling is problematic
  - AMD's Geode is another example of a processor near the blurry boundary between general-purpose processors and microcontrollers.

EMBEDDED PROCESSORS

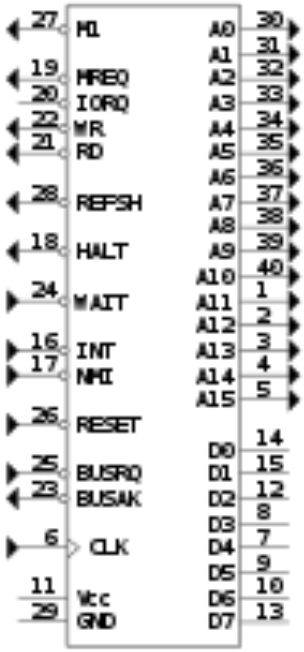
- **Microcontrollers**
  - The **Motorola 6800** and Intel 8080 are 8-bit microcontrollers that appeared on the market in 1974. Descendants of these architectures survive today, for example in the form of the Freescale 6811
  - The Zilog Z80 is a fully-compatible descendant of the 8080 that became one of the most widely manufactured and widely used microcontrollers of all time. A derivative of the Z80 is the Rabbit 2000 designed by Rabbit Semiconductor
  - The Intel 8051 is an 8-bit microcontroller developed by Intel in 1980. The 8051 ISA is today supported by many vendors, including Atmel, Infineon Technologies, Dallas Semiconductor, NXP, ST Microelectronics, Texas Instruments, and Cypress Semiconductor
  - The Atmel AVR 8-bit microcontroller, developed by Atmel in 1996, was one of the first microcontrollers to use on-chip flash memory for program storage. Although Atmel says AVR is not an acronym, it is believed that the architecture was conceived by two students at the Norwegian Institute of Technology, Alf-Egil Bogen and Vegard Wollan, so it may have originated as Alf and Vegard's RISC



Motorola 6800

EMBEDDED PROCESSORS

- **Microcontrollers**
  - The Motorola 6800 and Intel 8080 are 8-bit microcontrollers that appeared on the market in 1974. Descendants of these architectures survive today, for example in the form of the Freescale 6811
  - The **Zilog Z80** is a fully-compatible descendant of the 8080 that became one of the most widely manufactured and widely used microcontrollers of all time. A derivative of the Z80 is the Rabbit 2000 designed by Rabbit Semiconductor
  - The Intel 8051 is an 8-bit microcontroller developed by Intel in 1980. The 8051 ISA is today supported by many vendors, including Atmel, Infineon Technologies, Dallas Semiconductor, NXP, ST Microelectronics, Texas Instruments, and Cypress Semiconductor
  - The Atmel AVR 8-bit microcontroller, developed by Atmel in 1996, was one of the first microcontrollers to use on-chip flash memory for program storage. Although Atmel says AVR is not an acronym, it is believed that the architecture was conceived by two students at the Norwegian Institute of Technology, Alf-Egil Bogen and Vegard Wollan, so it may have originated as Alf and Vegard's RISC



Zilog Z80

EMBEDDED PROCESSORS

- **Microcontrollers**
  - Many 32-bit microcontrollers implement some variant of an **ARM** instruction set, developed by ARM Limited. ARM originally stood for Advanced RISC Machine, and before that Acorn RISC Machine, but today it is simply ARM. Processors that implement the ARM ISA are widely used in mobile phones to realize the user interface functions, as well as in many other embedded systems. Semiconductor vendors license the instruction set from ARM Limited and produce their own chips.
  - Other notable embedded microcontroller architectures include
    - the Motorola **ColdFire** (later the Freescale ColdFire)
    - the Hitachi **H8** and **SuperH**
    - the **MIPS** (originally developed by a team led by John Hennessy at Stanford University)
    - the **PIC** (originally Programmable Interface Controller, from Microchip Technology)
    - the **PowerPC** (created in 1991 by an alliance of Apple, IBM, and Motorola)
    - **RISC-V** (Open Source)