

## 1 Course Introduction

See the syllabus at <http://www.devlang.com/cse325>.

## 2 Introduction to Embedded Systems

[Ref: [Wikipedia: Embedded System](#)]

### 2.1 What is an Embedded System?

[Ref: HEATH, p. 2] "There are many definitions for this but the best way to define it is to describe it in terms of what it is not and with examples of how it is used. An embedded system is a **microprocessor-based system** that is built to control **a function** or range of functions and is **not designed to be programmed by the end user** in the same way that a PC is. Yes, a user can make choices concerning functionality but cannot change the functionality of the system by adding/replacing software. ... An embedded system is designed to **perform one particular task**."

[Ref: NOER, p. 5] "An embedded system is an **applied computer system**, as distinguished from other types of computer systems such as personal computers or supercomputers. ... the definition is fluid and difficult to pin down... Embedded systems are **more limited in hardware and/or software functionality** than a PC ... An embedded system is designed to **perform a dedicated function** ... An embedded system is a computer system with **higher quality and reliability requirements** than other types of computer systems."

[Ref: VAHID, p. 1-1] "... they are **embedded within larger electronic devices**, repeatedly carrying out a **particular function**, often **going completely unrecognized by the device's user**. Creating a precise definition of such embedded systems... is not an easy task ... An embedded system is **nearly any computing system other than a desktop, laptop, or mainframe computer**."

[Ref: WHITE, p. 1] "Embedded systems are different things to different people. To someone who has been working on servers, an application developed for a phone is an embedded system. To someone who has written code for tiny 8-bit microprocessors, anything with an operating system isn't very embedded. ... **embedded systems are things like microwaves and automobiles that run software but aren't computers**."

[Ref: BARR, pp. 1-2] "An embedded system is a combination of **hardware and software**... designed to **perform a specific function**. Frequently, an embedded system is a component **within some larger system**. If an embedded system is designed well, the existence of the processor and software **could be completely unnoticed by a user of the device**."

Combining these definitions we can come up with this definition:

*An ES is characterized by,*

- *It is a microprocessor- or microcontroller-based computer system.*
- *It is programmed for a specific fixed function and that function cannot be changed by the end user.*
- *It has or consumes fewer resources—e.g., memory, processor power—than a typical computer system.*
- *Many ES's must be highly reliable, highly available, and fail-safe, e.g., those in the medical or avionics field.*
- *An embedded system is often just one of many component of a larger system.*
- *The user of the ES is often not aware that it is a computer-based system, e.g., my coffee maker or a microwave.*

### 2.2 Where are Embedded Systems Found?

In just about any device that is controlled by electricity.

#### In the Home

*Appliances:* Refrigerators, Washers, Dryers, Microwaves, Coffee makers, Ovens. *Entertainment:* TV's, DVD's, Stereo's, DVR's, Cable boxes, Remote controls, Camcorders, Cameras, Wireless routers, MP3 players. *Other:* Security systems, Garage door openers, Alarm clocks, Exercise equipment, Toys, Thermostats.

### In the Office

Fax machines, Copiers, Laser printers, Digital telephones, IVR systems, Security systems, Printers, Environmental monitoring, HVAC control.

### In the Automobile

Electronic fuel ignition, Environmental monitoring, ABS, Instrumentation, Security systems, Transmission control, Air bag system, Climate control, Keyless entry, GPS navigation, Communication, Entertainment system, Active suspension.

### Literally, Almost Everywhere

ATM's, Airplanes, Trains, Medical devices, Industrial control, Satellites, Aerospace, The list goes on and on ... and on.

[Ref: <http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code>, 2009]

- The avionics in the F-22 Raptor—the current US Air Force front-line jet fighter—contains about 1.7 million LOC.
- The F-35 Joint Strike Fighter will contain about 5.7 million lines of code.
- Boeing's 787 Dreamliner contains about 6.5 million lines of code<sup>1</sup>.
- A high-end BMW or Mercedes will have around 100 million LOC.
- The radio and navigation software in the S-Class Mercedes requires over 20 million LOC and the car contains as many embedded microprocessors as the new Airbus A380 (excluding the plane's inflight entertainment system).
- Software and ES hardware can reach 35 to 40 per cent of a new car, with software development contributing about 13 to 15 per cent of the cost.

## 2.3 ES Characteristics

[Ref: <http://www.jopdesign.com/teaching/EmbeddedIntro.pdf>, Schoeberl, M., 2006]

### ES's are dedicated to a specific task [Ref: BERGER pp. 8–9; NOER p. 5; VAHID p. 1-1]

Embedded microprocessors are dedicated microprocessors. They are programmed to perform one, and only one, task. The microprocessor that controls a pacemaker does not, should not, and never will, function as a word processor.

Is a smart phone an ES? Personally, I do not consider smart phones to be ES's because they are essentially a general-purpose computer. They can be programmed to perform different tasks and the software is easily changeable.

### ES's are cost-sensitive [BERGER pp. 8, 10]

In many devices, the cost of the microprocessor, related hardware, and software is the bulk of the cost of the system (including development and manufacturing). Saving a few cents on a part can result in substantial savings over the product's lifetime.

For example, consider two designs A and B for an device containing an ES. Production volume is estimated to be 100,000 units per year, with a product lifetime of 5 years. Retail cost is \$100. The parts for design A are \$7.37<sup>2</sup> and for design B, \$7.36, and the manufacturing costs are otherwise identical, say \$12, so design A's manufacturing cost will be \$19.37 and design B's, \$19.36. Research and development costs for both designs are the same. How much savings would design B lead to over design A?

Number of units manufactured over the product lifetime =  $100,000 \times 5 = 500,000$ .

Manufacturing cost for design A =  $500,000 \times \$19.37 = \$9,685,000$

Manufacturing cost for design B =  $500,000 \times \$19.36 = \$9,680,000$

Gross profit =  $500,000 \times \$100 = \$50,000,000$

Net profit for design A =  $\$50,000,000 - \$9,685,000 = \$40,315,000$

Net profit for design B =  $\$50,000,000 - \$9,680,000 = \$40,320,000$

1 In contrast, the NASA space shuttle's computer systems were relatively old-fashioned and underpowered, with the primary flight control software consisting of just 400,000 LOC, running on a system with 1 MB of memory and a processor running at 1.4 MIPS.

[Ref: [http://www.nasa.gov/mission\\_pages/shuttle/flyout/flyfeature\\_shuttlecomputers.html](http://www.nasa.gov/mission_pages/shuttle/flyout/flyfeature_shuttlecomputers.html)]

2 Similar to an iPhone, which of course, retails for about a klockzillion dollars.

Therefore, design B will lead to an additional profit of  $\$40,320,000 - \$40,315,000 = \$5,000$ . \$5K may not seem like a lot, but realize that for every penny we save in design B, we earn an additional \$5,000 over the lifetime of the product. If design A uses a microprocessor which costs \$3.53 and the engineers realize they can achieve the same product requirements by using a \$1.17 microprocessor (less powerful, but powerful enough to meet the requirements) in design B, the overall savings would be  $(\$3.53 - \$1.17) \times \$5000 = \$1,180,000$ , and that would pay *your* salary for a year or two, thus keeping you employed.

This example leads to an important point to understand when choosing a microprocessor for a design: *a design goal should always be to consider for adoption the least expensive microprocessor that will get the job done*<sup>3</sup>.

### ES's often must meet real-time constraints [Ref: BERGER pp. 8, 10; VAHID p. 1-2; WHITE p. 1]

A **real-time system** is one that must react to external events **when they happen** or as shortly thereafter as possible. ES's are generally real-time systems. Real-time events can be partitioned into two broad categories based on the consequences of missing a deadline:

**Soft real-time:** A soft real-time event is one that if the ES does not react to it within the designed time constraint it will only temporarily degrade the quality of the system. Example: an audio or video decoder that misses an event which causes a temporary degradation in the quality of the audio or video signal.

**Hard real-time:** A hard real-time event is one that the system must react to within the designed time constraints and if it does not, it will lead to complete system failure. In some hard real-time systems, this complete system failure may be catastrophic, i.e., result in loss of property, life, or cause great damage to the environment. Examples: the air bag system in an automobile (the system must respond to a collision event within 10-50 milliseconds), a pacemaker, missile defense system.

### Many ES's either use no OS or if they do, use a Real-Time Operating System (RTOS) [BERGER pp. 9–10]

Many ES's are, relatively speaking, simple enough that they do not require a full-blown operating system. In these cases, a simple task-scheduling **executive**<sup>4</sup> may be sufficient. The job of the executive is to run tasks at the required rates.

A Real-Time Operating System<sup>5</sup> (RTOS) provides many of the features of a traditional operating system, including task scheduling, resource sharing, and interprocess communication. In addition, a RTOS is specifically designed to react to soft and hard real-time events. Windows (and Linux for that matter<sup>6</sup>) are not RTOS's. Windows is the antithesis of real-time because I frequently see the stupid hourglass icon telling *me* to wait on it!

One characteristic of a RTOS is that it provides **consistency** concerning the time it takes to recognize an event and complete the event's task handler; variability in this time is referred to as **jitter**, and for hard real-time events, zero jitter is the goal (which cannot realistically be met, so in practice the system is designed so the maximum delay is less than some epsilon, where epsilon is relatively small).

RTOS's provide advanced scheduling algorithms for running tasks at various rates and ensuring that higher-priority tasks are never **preempted**<sup>7</sup> by lower-priority tasks.

3 Although in some cases the microprocessor selection is limited to a certain manufacturer because of company policy or because you already have all of the development tools you need for a certain microprocessor and switching families or vendors would result in having to purchase all new development tools.

4 <http://www.drdobbs.com/a-simple-real-time-executive/184402613>

5 [http://en.wikipedia.org/wiki/Real-time\\_operating\\_system](http://en.wikipedia.org/wiki/Real-time_operating_system)

6 Embedded Linux is a RTOS, see [http://en.wikipedia.org/wiki/Embedded\\_Linux](http://en.wikipedia.org/wiki/Embedded_Linux).

7 A preemptive system is one in which a lower priority task may be interrupted by a higher priority task which runs to completion before control returns to the lower priority task. Some ES's use preemptive schedulers and some do not, see [http://en.wikipedia.org/wiki/Preemption\\_\(computing\)](http://en.wikipedia.org/wiki/Preemption_(computing)).

Popular RTOS's (a more comprehensive list of RTOS's can be found at <sup>8</sup>)

INTEGRITY	<a href="http://en.wikipedia.org/wiki/Integrity_(operating_system)">http://en.wikipedia.org/wiki/Integrity_(operating_system)</a>
LynxOS	<a href="http://en.wikipedia.org/wiki/LynxOS">http://en.wikipedia.org/wiki/LynxOS</a>
OSE	<a href="http://en.wikipedia.org/wiki/Operating_System_Embedded">http://en.wikipedia.org/wiki/Operating_System_Embedded</a>
QNX	<a href="http://en.wikipedia.org/wiki/QNX">http://en.wikipedia.org/wiki/QNX</a>
VxWorks	<a href="http://en.wikipedia.org/wiki/VxWorks">http://en.wikipedia.org/wiki/VxWorks</a>
Windows CE	<a href="http://en.wikipedia.org/wiki/Windows_CE">http://en.wikipedia.org/wiki/Windows_CE</a>
Embedded Linux	<a href="https://en.wikipedia.org/wiki/Linux_on_embedded_systems">https://en.wikipedia.org/wiki/Linux_on_embedded_systems</a>

Embedded Linux is not exactly a RTOS, but it is commonly used in ES's. According to Wikipedia, the primary disadvantage of embedded Linux over a RTOS is the relatively large memory footprint and a complex device drivers framework (a device driver is software that interfaces to a hardware device; essentially, it becomes part of the OS if you tweak the right things).

A program running in an ES which does not use a RTOS is said to be running on **bare metal**. In CSE325, we will not use a RTOS so our programs will run on bare metal.

### Software failure can be catastrophic [BERGER pp. 9, 11; NOER p. 6; WHITE p. 1]

We touched on this when we discussed real-time event response. However, catastrophic failures can manifest themselves in nonreal-time ways. Six notorious failures are discussed below.

**Therac-25** [<http://en.wikipedia.org/wiki/Therac-25>; [http://courses.cs.vt.edu/~cs3604/lib/Therac\\_25/Therac\\_1.html](http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html)]

The Therac-25 was a radiation therapy machine which was involved in six accidents between 1985 and 1987 in which patients were mistakenly given massive overdoses of radiation. The root cause was a certain sequence of keystrokes entered within an 8 second interval. If I remember correctly, a couple of patients died from their injuries and some of the others were horribly burned leading to loss of function of the affected area.

**Ariane 5 Flight 501** [[http://en.wikipedia.org/wiki/Ariane\\_5\\_Flight\\_501](http://en.wikipedia.org/wiki/Ariane_5_Flight_501)]

Ariane 5 was (may still be) a French rocket designed to carry satellites into space. On 4 June 1996 flight 501 was launched and 37 seconds thereafter it began to veer off course (heading somewhere near downtown Paris) and self-destructed. The root cause was a data conversion from a 64-bit floating point value to a 16-bit signed integer value which overflowed. The destroyed rocket and its payload was valued at \$500 million.

*Lesson:* A 64-bit IEEE 754 floating point value can range from approximately  $\pm 10^{308}$ . A signed two's complement 16-bit integer can represent values in the range -32768 to 32767. If my math is correct,  $10^{308}$  is a bit larger than  $32767^9$ . In a mission-critical application, you would be a fool to not check for overflow when performing such a conversion.

**Soviet Phobos 1 Mars Probe** [[http://en.wikipedia.org/wiki/Phobos\\_program](http://en.wikipedia.org/wiki/Phobos_program)]

Phobos was a Soviet-led program which launched two probes, Phobos 1 and 2, to study Mars and its moons. Phobos 1 launched on 7 July 1988 and operated normally until controllers lost communication with the probe. The root cause was a bug in software uploaded on 29 Aug which deactivated the attitude thrusters. This caused Phobos 1 to lose its lock on the Sun, which was critical for aligning the solar arrays to charge the batteries, and eventually the batteries failed and the probe was lost. Interestingly, the software which turned off the attitude control was intended to only be used during testing on the ground. However, the software was coded in PROMS, and so removing the test code before launch would have required removing and replacing the entire computer system. Because of the time-pressure from the impending launch, a decision was made to leave the code intact, knowing that it would never be activated. During the software upload on 29 Aug, a single-character error resulted in the offending code being executed.

*Lesson:* See <sup>10</sup>.

<sup>8</sup> [http://en.wikipedia.org/wiki/List\\_of\\_real-time\\_operating\\_systems](http://en.wikipedia.org/wiki/List_of_real-time_operating_systems)

<sup>9</sup> Approximately  $3^{303}$  times larger.

<sup>10</sup> [http://en.wikipedia.org/wiki/Murphy's\\_law](http://en.wikipedia.org/wiki/Murphy's_law)

**Apollo 11 Lunar Module** [<http://www.hq.nasa.gov/office/pao/History/alsj/a11/a11.1201-pa.html>]

During the descent of the Apollo 11 lunar module carrying Neil Armstrong and Buzz Aldrin to the moon, at 6000 feet above the surface the navigation and guidance computer began to issue code 1201 and 1203 alarms. These alarms indicated executive overflows, meaning the guidance computer could not complete all of its tasks in real-time and had to postpone some of them. The root cause was a radar switch that was in the wrong position (due to an error in the astronaut's checklist manual) which caused a flood of spurious data, thus leading to the executive failure. Armstrong was forced to semi-automatically pilot the lander while attempting to locate a boulder-free area and finally landed with just 25 seconds of fuel remaining.

*Lesson:* The software essentially functioned as designed, and the root cause of the failure was the erroneous checklist manual. Don't let the clueless write manuals.

**Mars Climate Orbiter** [<http://mars.jpl.nasa.gov/msp98/news/mco991110.html>]

This probe was launched by NASA on 11 Dec 1998 to study the Martian climate. On 23 Sep 1999 communication was lost as the probe went into orbital insertion. Ultimately, the probe entered the atmosphere at an incorrectly low altitude and burned up. The root cause was ground-based software producing an output in Imperial units (or British units) instead of the metric units which the probe was expecting.

*Lesson:* The United State is—along with those bastions of freedom, Burma and Liberia—one of only three countries in the world that does not officially mandate use of the metric system. When I was a kid in elementary school in the early 70's, some of our days in mathematics were filled with exercises in converting imperial units to metric units and learning about the metric system as the USA was, finally, in the mid 70's on the verge of moving to the metric system. That was 40 years ago and where are we at now? Soda now comes in 2 liter bottles instead of the old 32 oz bottles. Quite frankly, I don't think America is, on the whole, composed of enough citizens of above-dullard intelligence that we will ever be able to convert<sup>11</sup>.

**Patriot Missile Failure** [<http://www.ima.umn.edu/~arnold/455.f96/disasters.html>]

During the Gulf War in 1991, an American Patriot missile system failed to intercept an incoming Iraqi Scud missile which struck an Army barracks killing 28 people. The root cause was an incorrect calculation of time due to a rounding error in the fraction  $1/10^{12}$ . The Patriot missile system was not designed to be left continuously operating without rebooting, and over the course of the several days that it was, the resulting roundoff error was magnified leading to the timing error.

*Lesson:* It is generally not a good ideal in safety-critical systems to try to work around roundoff errors. Also, one should try to avoid floating-point data types and operations at all costs.

The Risk's Digest<sup>13</sup> is an interesting read on various software failures.

**Power constraints [BERGER 9, 11; VAHID 1-1; WHITE 1]**

ES's often run on batteries or in environments where something like this,



is not feasible (that is a huge heat sink and a fan sitting on top of a microprocessor, cooling it down). In some ES's the batteries are nonrechargeable (because they are not easily accessible) and the system must run for months or years on battery power.

<sup>11</sup> Imagine the carnage that would take place on the roads. And I'm being optimistic.

<sup>12</sup>  $1/10$  has a nonterminating binary expansion.

<sup>13</sup> <http://catless.ncl.ac.uk/Risks/>

**Environmental conditions [BERGER pp. 9, 12]**

ES's are found miles underwater, on mountaintops, underground, and in space. Some may need to operate in temperatures ranging from -200 degrees C to 100 degrees C. They must be designed to handle moisture, pressure, acceleration, high g-forces, and other factors that no desktop PC will ever experience.

**Fewer resources than a desktop system [BERGER pp. 9, 13; WHITE pp. 1, 4]**

I got a new office computer this year. It's a Dell latitude laptop with a 2+ GHz Intel i7 processor, 16 GB of RAM, and a 256 GB solid state drive. It has a 14" touch LCD, about a thousand keys on the keyboard, a touchpad, and a host of external interfaces, e.g., ethernet, USB, firewire. It is, in no way, typical of an ES.

Some ES's are built around 4- or 8-bit microprocessors<sup>14</sup>. Some operate with as few as 16 to 256 bytes<sup>15</sup> of RAM, and in many cases, 64 or 128 KB of RAM would be considered a significant amount of memory. Processor speeds may be as low as 1 MHz. Fancy screens and user interfaces are a rarity.

**Cost** and **performance** are major criteria for evaluating parts going into an ES, with the goal of minimizing cost and maximizing performance, but only to the point that the system will meet its requirements. Sure, you could put a 64-bit ARMv8-A chip running at 1.4 GHz with 512 KB of SRAM in a coffee maker, and no doubt, it would make a decent cup of coffee, but it wouldn't be very cost-effective when a 4-bit 0.5 MHz microcontroller with 32 bytes of SRAM and 1 KiB of ROM would do the same job and for a lot less in dollars.

**Code stored in ROM [BERGER pp. 9, 13]**

ES's typically do not contain hard drives, CD or DVD drives, or other forms of permanent storage, other than some form of ROM (flash is dominant today). We will talk about memory and ROM later in the course, but realize that when an ES is powered up, the code must be present in ROM and ready to run. Often the code, or only part of it, will be copied from ROM to RAM from where it will be executed, but of course, RAM is volatile so the code and data must be stored in a permanent form of memory when the system is powered down.

**ES microprocessors have special debug circuitry [BERGER pp. 9, 15]**

Because of the limitations of ES processors, most newer chips intended for ES's are manufactured with debugging circuitry built into the chip. Motorola developed the first microprocessors with such circuitry in their 683xx processor family; they called this Background Debug Mode<sup>16</sup> (BDM). ARM chips employ the CoreSight debug and trace technology<sup>17</sup> using either a JTAG<sup>18</sup> or Serial Wire Debug<sup>19</sup> (SWD) interface.

**Require special development tools [BERGER pp. 9, 14; WHITE pp. 1, 3]**

ES's can be more difficult to debug than a program for a PC—because of the lack of hardware resources and real-time requirements—so there are various debugging tools and techniques that are commonly used. As mentioned above, modern microcontrollers for the ES market contain specialized, dedicated debug hardware built into the processor. The software debugger runs on a PC and communicates with the microcontroller, usually over USB.

Freescale puts an OpenSDA<sup>20</sup> debug interface on their Kinetis development boards, including the FRDM-KL46Z board we will use in this course. OpenSDA consists of a Kinetis K20 MCU which interfaces the PC-based software debugger to the target Kinetis MCU. OpenSDA also provides the ability to program the flash of the target MCU. Freescale provides a customized version of Eclipse called CodeWarrior which includes the C/C++ IDE, compiler, assembler, linker, software libraries, and other tools.

14 The Renesas  $\mu$ PD67/68/69 family: <http://documentation.renesas.com/doc/DocumentServer/U14935EJ2V1DS00.pdf>. The venerable Intel 8-bit 8051 architecture, [https://en.wikipedia.org/wiki/Intel\\_MCS-51](https://en.wikipedia.org/wiki/Intel_MCS-51), is still widely used.

15 That's not a typo; I meant bytes. For example, the PIC10F20x family, <http://ww1.microchip.com/downloads/en/DeviceDoc/40001239E.pdf>, which comes in two flavors with a whopping 16 or 24 bytes of SRAM.

16 [http://en.wikipedia.org/wiki/Background\\_Debug\\_Mode\\_interface](http://en.wikipedia.org/wiki/Background_Debug_Mode_interface); [http://www.freescale.com/files/microcontrollers/doc/app\\_note/AN3335.pdf](http://www.freescale.com/files/microcontrollers/doc/app_note/AN3335.pdf)

17 <http://www.arm.com/products/system-ip/debug-trace/>

18 <http://en.wikipedia.org/wiki/JTAG>; [http://www.corelis.com/education/JTAG\\_Tutorial.htm](http://www.corelis.com/education/JTAG_Tutorial.htm)

19 <http://www.arm.com/products/system-ip/debug-trace/coresight-soc-components/serial-wire-debug.php>

20 [http://www.freescale.com/webapp/sps/site/prod\\_summary.jsp?code=OPENSDA](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=OPENSDA)



ARM has developed the mbed<sup>21</sup> suite of web-based development tools for the ARM architecture. mbed provides a free C/C++ software library (the mbed SDK<sup>22</sup>), the mbed Hardware Development Kit<sup>23</sup>, and the mbed C/C++ IDE<sup>24</sup>. The platform is designed for rapid prototyping of produces based on ARM microcontrollers. We will not be using mbed in CSE325, but you might want to play around with it if you are interested. Note, however, that this would require flashing new firmware to the K20 debug chip on the FRDM-KL46Z board that would support the mbed tools (the firmware currently contains code to support the OpenSDA protocol).

### High availability

Some ES's must run continuously for months or years without being powered down, or if they are powered down, it may be for only a very short time—e.g., consider an implanted medical device such as a pacemaker or insulin pump.

## 3 Overview of ES Components

### 3.1 Processors

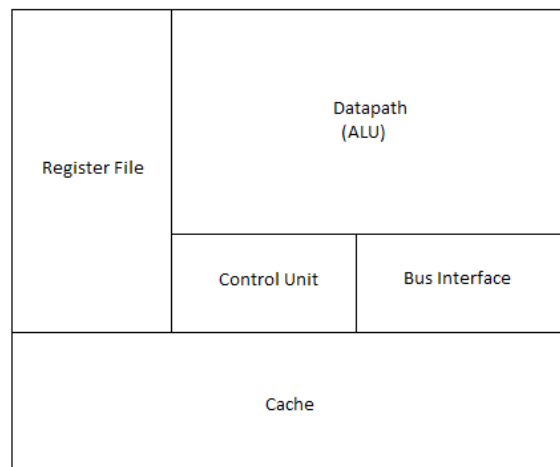
You all know what a microprocessor is so I won't spend anytime discussing that. Suffice it to say, it wouldn't be an ES without a processor of some sort.

#### 3.1.1 Processor-Taxonomy

There are several different types of processors used in ES's.

#### General-Purpose Processor — Microprocessor

[Ref: VAHID p. 1-5] A **general-purpose (GP) microprocessor**<sup>25</sup> is designed to be used in a wide variety of designs (laptops, desktop systems, supercomputers, pad computers, ES's) with a primary goal being to maximum processor sales for the manufacturer (which is accomplished by selling the chips in large volume; hence they must be very versatile). Intel's Pentium, Celeron, Atom, Core, Core 2, and Xeon chips and AMD's Athlon, Sempron, Turion, Opteron, and Phenom chips—which are used in almost all desktop, laptops, and supercomputers—are examples of general-purpose processors. Major components,



GP processors are not commonly used in ES's because in many cases they are overkill, i.e., offer more performance than the system requires, consume too much power, and/or also cost more than alternative processors. They also lack many of the peripheral components that ES's rely on, and although these components can be added to the design, they will be external to the processor and thus increase the size of the printed circuit board (PCB).

<sup>21</sup> mbed: <https://developer.mbed.org/>

<sup>22</sup> mbed SDK: <https://developer.mbed.org/handbook/mbed-SDK>

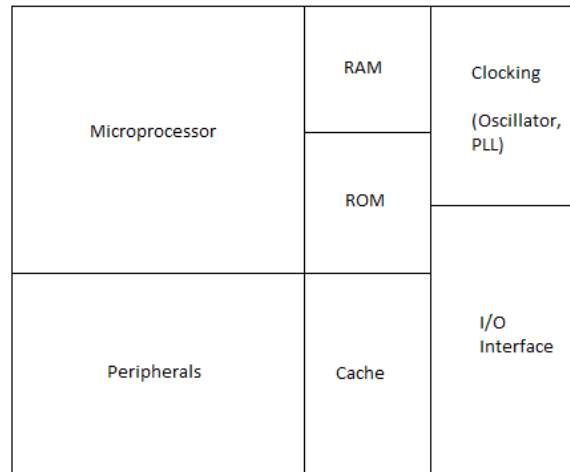
<sup>23</sup> mbed HDK: <https://developer.mbed.org/handbook/mbed-HDK>

<sup>24</sup> mbed Compiler: <https://developer.mbed.org/handbook/mbed-Compiler>

<sup>25</sup> <http://en.wikipedia.org/wiki/Microprocessor>

### General-Purpose Processor — Microcontroller

Microcontrollers are specifically designed for ES applications and 4-, 8-, 16-, 32-, and 64-bit microcontrollers are widely used. (FYI, Texas Instruments developed the world's first microcontroller, the TMS 1000<sup>26</sup>, in 1971).



Microcontrollers have numerous advantages over microprocessors in ES's, [Ref: BERGER, p. 34]

- The system will generally be of lower cost because one part replaces many.
- The system may use a smaller PCB (due to the reduced parts count).
- The system may be more reliable because fewer packages means fewer solder joints and fewer interconnects.
- A microcontroller may offer better performance than a microprocessor because all signals stay on the chip.
- A microcontroller design may radiate less RF energy (fast signals on a PCB radiate RF energy).

### Single-Purpose Processor

[Ref: VAHID, p. 1-6] A single-purpose processor is one designed to execute exactly one program (these types of processors are sometimes referred to a **co-processors** (because they are part of a system containing a microprocessor or microcontroller and the microcontroller offloads some of the work to the co-processor) or **accelerators** (because they are employed to speed up a compute-intensive task and/or to offload the task from the microcontroller). The basic difference between a single-purpose and general-purpose processor is that where a general-purpose processor has an execution unit designed to execute a wide variety of instructions, a single-purpose processor will have an execution unit designed to execute a specific set of instructions making up a program (think of it like a software program implemented in hardware). An example would be an MP3 codec chip<sup>27</sup>. Advantages of designing a single-purpose processor include,

- May achieve better performance, power consumption, and size compared to a GP processor.
- In large volumes, unit costs<sup>28</sup> may be low.

Disadvantages include,

- Nonrecurring engineering (NRE) cost<sup>29</sup> may be high compared to a commercial off-the-shelf (COTS) chip<sup>30</sup>.
- Flexibility is low, e.g., if a bug is found or new features are added, the chip may need to be redesigned and retested.
- In small quantities, unit cost may be high.

<sup>26</sup> [http://en.wikipedia.org/wiki/TMS\\_1000#TMS\\_1000](http://en.wikipedia.org/wiki/TMS_1000#TMS_1000)

<sup>27</sup> VLSI Solution VS1011e MP3 Decoder Chip. <http://www.vlsi.fi/fileadmin/datasheets/vs1011.pdf>

<sup>28</sup> Manufacturing cost.

<sup>29</sup> NRE cost is what it costs the company to design and develop the chip. It is a one-time expense, hence, nonrecurring, although it could be quite large.

<sup>30</sup> A COTS chip is one that someone else has designed, manufactured, and is selling. They have paid the NRE and unit costs for the chip and are attempting to gain back these costs and to make a profit by selling the chip in volume.



### Application-Specific Instruction Set Processor (ASIP)

[Ref: VAHID, p. 1-7] The goal of using an ASIP is to try to gain the advantages of using both a general-purpose and special-purpose processor without the disadvantages of each. Whereas a single-purpose processor is not really programmable (the hardware *is* the program and these can be very expensive to design and manufacture), an ASIP (less expensive to design and manufacture) can be reprogrammed via software. But, unlike a general-purpose processor (which most likely implements many instructions that are unneeded), an ASIP has a limited set of instructions (a custom datapath) that are applicable to implementing certain types of programs.

Probably the most common type of ASIP is a Digital Signal Processor (DSP)<sup>31</sup> which is a processor customized for digital signal processing applications (think audio and video coding and decoding). One very common digital signal processing application is in cell phones, where an analog signal (your voice) must be quickly digitized (converted into digital format) and processed, or coming the other direction, the digital signal coming from the cell tower must be converted into analog format and sent to the speaker.

ASIP advantages include,

- Will achieve better performance, power, and size than a GP processor.
- More programmable flexibility than a SP processor.

Disadvantages,

- NRE and unit costs may be very high (but not as high as a SP processor).
- Special software tools, such as a custom compiler, may need to be built if they are not available from other vendors.
- If a special compiler is not available, one can always code a large part of the application in assembly language, but this slows down software development.

### 3.1.2 Processor Implementation

[Ref: VAHID p. 1-8] Every processor must eventually be implemented in silicon as an integrated circuit<sup>32</sup> (IC) or chip. At an abstract view, semiconductor chips consist of three layers on top of a silicon substrate<sup>33</sup>,

Wiring
Logic Gates
Transistors
Silicon

### Full-Custom (VLSI) Design

[VAHID p. 1-8] In a full-custom design (Very Large Scale Integration or Ultra Large Scale Integration<sup>34</sup>) the engineer designs the chip from the bottom silicon layer up and then sends the design to an IC fab<sup>35</sup> for fabrication, e.g., Intel, TI, IBM, Samsung and many other companies will do this for you. This is exactly the way chip makers such as AMD and Intel design their microprocessor IC's.

Advantages,

- Will yield the best performance, power consumption, and size of any implementation.

31 [http://en.wikipedia.org/wiki/Digital\\_signal\\_processor](http://en.wikipedia.org/wiki/Digital_signal_processor)

32 [http://en.wikipedia.org/wiki/Integrated\\_circuit](http://en.wikipedia.org/wiki/Integrated_circuit)

33 The future is in 3D integrated circuits which stack multiple layers (wiring, gates, transistors) on top of each other.

34 VLSI circuits consist of 100,000+ transistors. The term ULSI is used by some to refer to IC's that integrate 1,000,000+ transistors.

35 [http://en.wikipedia.org/wiki/Fab\\_\(semiconductors\)](http://en.wikipedia.org/wiki/Fab_(semiconductors)). A company that manufactures their own chips as well as fabricating chips for customers is known as a foundry. A company owning a fab that does not design their own chips, but rather, operates the fab for implementing the designs of fabless semiconductor companies is called a pure-play foundry. The largest of these is the Taiwan Semiconductor Manufacturing Company (TSMC).

Disadvantages,

- Very high NRE cost (design) and long turnaround times (weeks to months) before the chip is manufactured.
- Only cost-effective in very high volume applications.
- Post-production bugs can be very expensive to fix.

### Semi-Custom Application Specific Integrated Circuit (ASIC)

[VAHID p. 1-8] A **standard cell ASIC** is a design in which the lower layer (the transistor layer) is fully or mostly built and the engineer designs the middle (logic gate) and upper (wiring) layers. In a **gate array ASIC**<sup>36</sup> the transistor and gate layers are prebuilt and the engineer simply designs the wiring layer. ASIC's are widely used in ES's (to implement single-purpose and ASIP designs) as they still provide very good performance, power, and size, with less NRE cost than a full-custom design. But, they are still costly to design (although gate array ASIC's are less costly than standard cell ASIC's), have relatively long turnaround times, and design mistakes can still be very costly to correct.

### Programmable Logic Device (PLD)

[VAHID p. 1-9] In a PLD all of the layers (transistor, logic gate, wiring) exist. Programming consists of creating or destroying connections between wires that connect gates to implement the design. Unlike ASIC's, which must be fabricated by an IC fab, a PLD is user-programmable, i.e., the design does not have to be sent to a fab.

Field Programmable Gate Arrays<sup>37</sup> (FPGA's) are a particular type of programmable logic device that have become extremely popular since the early 1990's. They are essentially user-programmable gate array ASIC's. You will learn all about these in CSE320.

Advantages of PLD's and FPGA's are low NRE costs, user-programmability, quick turnaround time, and flexibility. Disadvantages are that they will achieve lower performance than an VLSI or ASIC design, will be larger, will consume more power, and will have higher unit costs. Often, an FPGA will be used during prototyping to prove the design before final implementation in VLSI or an ASIC.

## 3.2 Storage

### 3.2.1 Read Only Memory (ROM)

ROM stores bits (instructions and data) permanently when the system is powered off. Various types of ROM have been developed over the years, and we will discuss memory technology in more detail later in the course, so this is just a high-level overview.

#### Mask ROM

A ROM designed and manufactured at the transistor-level. A fab will build these for you.

#### Programmable ROM (PROM)

Similar to a mask ROM, but user-programmable. Most PROM's are one-time programmable.

#### Erasable Programmable ROM (EPROM)

A PROM that can be erased and reprogrammed multiple times. Erasure is performed by exposing the chip to UV light.

#### Electrically-Erasable Programmable ROM (EEPROM)

An EPROM that can be erased by applying current to the transistors.

#### Flash

Flash ROM is a form of EEPROM that is very similar, but also different, than traditional EEPROM.

---

<sup>36</sup> [http://en.wikipedia.org/wiki/Gate\\_array](http://en.wikipedia.org/wiki/Gate_array)

<sup>37</sup> [http://en.wikipedia.org/wiki/Field-programmable\\_gate\\_array](http://en.wikipedia.org/wiki/Field-programmable_gate_array)

### 3.2.2 Random Access Memory (RAM)

RAM stores bits (instructions and data) for a program which is currently being executed. The primary difference from ROM is that RAM will lose its contents when power is removed (RAM is volatile). There are as many (or more) types of RAM than there are ROM. RAM is broadly classified into two categories: static RAM (SRAM) and dynamic RAM (DRAM). The basic difference between the two is that SRAM is generally faster, less-dense (stores fewer bits per mm<sup>2</sup>), and more expensive than DRAM.

## 3.3 Peripherals

Peripherals are components that provide various functionality that is often required in an ES. These are called peripherals because historically the chips were located outside the microprocessor chip. Microcontrollers have peripherals embedded on the same die as the microprocessor and this is why microcontrollers have become so popular in the ES market.

### 3.3.1 Timers and Counters

All computer systems must be able to count and keep track of time. A typical microcontroller will contain several different types of timers and counters (timers and counters are closely related, as timers are implemented using counters).

#### Watchdog (Computer Operating Properly; COP) Timer<sup>38</sup>

[Ref: KL46-SRM SIM module, §3.4.10] A programmable countdown timer that must be "kicked" periodically to prevent the counter from reaching zero. If the counter reaches zero, the watchdog can be programmed to trigger a system reset. In a correct program, the watchdog timer will never expire, so these are used to reset the system when a bug manifests itself and causes the system to lock up or go off the deep end. We will probably not do anything with the COP in CSE325, other than to disable it at startup.

#### Programmable Interval Timer<sup>39</sup>

[Ref: KL46-SRM PIT module, Ch. 32] A programmable timer that is designed to generate interrupts<sup>40</sup> at a desired rate. We will study this timer module in CSE325.

#### Input Capture<sup>41</sup>

[Ref: KL46-SRM, TPM/PWM module, §31.4.4] A timer feature that can monitor an I/O pin for a change in signal. When the change occurs, the timer will store the value of the timer counter in a register and, if desired, generate an interrupt. We will use this module in CSE325.

#### Output Compare<sup>42</sup>

[Ref: KL46-SRM, TPM/PWM module, §31.4.5] A timer feature that can set or clear an I/O pin when the timer counter reaches a programmed value. Can be used to generate a periodic waveform, e.g., a square wave.

#### Pulse-Width Modulation (PWM)<sup>43</sup>

[Ref: KL46-SRM TPM/PWM module, Ch. 31] Generates a synchronous series of pulses on an output pin with programmable period and duty cycle. We will use this module in CSE325.

#### Real-Time Clock (RTC)<sup>44</sup>

[Ref: KL46-SRM Real-time clock module, Ch. 34] Maintains time of day and provides stopwatch and alarm functionality.

---

38 [http://en.wikipedia.org/wiki/Watchdog\\_timer](http://en.wikipedia.org/wiki/Watchdog_timer)

39 [http://en.wikipedia.org/wiki/Programmable\\_interval\\_timer](http://en.wikipedia.org/wiki/Programmable_interval_timer)

40 <http://en.wikipedia.org/wiki/Interrupt>

41 [http://en.wikipedia.org/wiki/Input\\_capture](http://en.wikipedia.org/wiki/Input_capture)

42 [http://en.wikipedia.org/wiki/Output\\_compare](http://en.wikipedia.org/wiki/Output_compare)

43 [http://en.wikipedia.org/wiki/Pulse-width\\_modulation](http://en.wikipedia.org/wiki/Pulse-width_modulation)

44 [http://en.wikipedia.org/wiki/Real-time\\_clock](http://en.wikipedia.org/wiki/Real-time_clock)

### 3.3.2 Input/Output and Communication

Communication among ES components, whether inter-chip, intra-chip (on the same or a different PCB), or with the external environment is an integral part of all ES's. We will spend quite a bit of time in CSE325 on communication, but for now here are few characteristics you should know:

- Communication speeds are measured in **bits per second** (bps), **kilobits per second** (Kbps), **megabits per second** (Mbps), **gigabits per second** (Gbps), or **terabits per second** (Tbps)<sup>45</sup>.
- A communication channel can be **synchronous** (clocked) or **asynchronous** (unclocked).
- The direction of data transfer in a communication channel can be \_\_\_\_\_ (one direction only), \_\_\_\_\_ (bidirectional, but in only one direction at a time), or \_\_\_\_\_ (simultaneously bidirectional).
- **Parallel** communication (or transfer) is the process of sending multiple bits simultaneously, while **serial** communication (or transfer) is the process of sending data sequentially, one bit after the other. There are advantages and disadvantages to each which we will discuss later in the course.

#### Discrete or General-Purpose I/O<sup>46</sup>

[Ref: KL46-SRM PORT and GPIO modules, Chs. 11, 42] All microcontrollers have pins which can be configured for input or output mode. The number of pins will vary from family to family and even within families. One of the most important selection criteria for a microcontroller is to make sure you select one that has an adequate number of I/O pins for your application. We will discuss the Kinetis MKL46Z GPIO module soon.

#### Universal Asynchronous Receiver Transmitter<sup>47</sup> (UART)

[Ref: KL46-SRM UART modules, Chs. 39-40] Handles the RS-232 serial I/O protocol in hardware. We will discuss this module in CSE325.

#### Inter-Integrated Circuit(I<sup>2</sup>C or IIC or I2C)<sup>48</sup>

[Ref: KL46-SRM I2C Module, Ch. 38] A standard serial I/O protocol commonly used in ES's. We will discuss this module.

#### Serial Peripheral Interface (SPI)<sup>49</sup>

[Ref: KL46-SRM SPI module, Ch. 37] A serial I/O protocol originally developed by Motorola, but now widely implemented by other chip vendors and widely used in ES's. We will discuss this module.

#### Ethernet<sup>50</sup>

A standard computer networking protocol designed for local area networks (LAN's). Ethernet is complicated as heck and the MKL46Z MCU does not have an ethernet module, so we will not discuss it.

#### Universal Serial Bus (USB)<sup>51</sup>

[Ref: KL46-SRM USBOTG module, Ch. 35] A standard serial I/O protocol widely used in the PC and electronics industries. USB is also complicated as heck so we will not discuss it.

#### Controller Area Network (CAN)<sup>52</sup>

An asynchronous bus protocol that is widely used in the automotive and industrial control markets. The MKL46Z module does not have a CAN controller so we will not discuss it.

45 Lower case b means bits; upper case B means bytes. Consequently, 100 Kbps is  $100 \times 2^{10}$  bits per second and 100 KBps is  $100 \times 2^{10}$  bytes per second (which is  $100 \times 2^{10} \times 2^3$  bits per second).

46 [http://en.wikipedia.org/wiki/General\\_Purpose\\_Input/Output](http://en.wikipedia.org/wiki/General_Purpose_Input/Output)

47 [http://en.wikipedia.org/wiki/Universal\\_asynchronous\\_receiver/transmitter](http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter)

48 <http://en.wikipedia.org/wiki/I%C2%B2C>

49 [http://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface\\_Bus](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)

50 <http://en.wikipedia.org/wiki/Ethernet>

51 [http://en.wikipedia.org/wiki/Universal\\_Serial\\_Bus](http://en.wikipedia.org/wiki/Universal_Serial_Bus)

52 [http://en.wikipedia.org/wiki/CAN\\_bus](http://en.wikipedia.org/wiki/CAN_bus)

### 3.3.3 Direct Memory Access (DMA)

[Ref: KL46-SRM DMA module, Ch. 23] A feature in many microprocessor and microcontroller systems which permits hardware devices other than the CPU to access system memory and transfer blocks of data from and to memory, with little processor interaction.

### 3.3.4 Interrupt Controller<sup>53</sup>

[Ref: ARM-M0+DUG, NVIC module, §4.2] An interrupt is an event—either hardware-generated or software-generated—that interrupts the microprocessor's normal execution sequence and causes it to execute the code of an interrupt service routine (ISR). Interrupts are very common and extremely useful in all computer systems, so we will spend quite a bit of time discussing the ARM Nested Vector Interrupt Controller (NVIC) module.

### 1.3.5 Analog-to-Digital<sup>54</sup> and Digital-to-Analog<sup>55</sup> Conversion

[Ref: KL46-SRM ADC module, Ch. 28; DAC module, Ch. 30] Various analog sensors or transducers can be used to interface with the external environment, e.g., a temperature sensor. Since the real world is analog (continuous; e.g., temperature, voltage, light intensity) but computers are digital, there must be a way to convert an analog signal to digital or vice versa. An A/D converter (or ADC) is a circuit which converts a continuous analog signal into a stream of digital values and a D/A converter or (DAC) performs the inverse operation. We will discuss the MKL46Z ADC and DAC modules in CSE325.

## 3.5 Software

Obviously, the processor in the ES executes instructions, and therefore the system is controlled by software. Overwhelmingly, the most popular ES programming language is \_\_\_\_\_. Other reasonably popular languages<sup>56</sup> are,

### C++

C++ compilers have advanced to the point where they can generate very efficient code so it *can* be used in ES's, although my hunch is that it would work better in 32- and 64-bit environments with a fair amount of memory and a RTOS. Interestingly, there is a dialect of C++, Embedded C++ or EC++<sup>57</sup>, which is a reduced-feature variant of C++ specifically designed for ES's, although it does not seem to be very popular.

### Assembly Language

Ah, my favorite programming language. Assembly is the language of choice when you want complete control over the code, e.g., when speed or space is of the utmost concern. It's also popular in ES's with few resources such as small 4- or 8-bit microprocessor with only a few bytes of RAM. Rarely would a nontrivial system today be written entirely in assembler, although certain time-critical parts may be. Another use of assembler is to perform low-level microprocessor operations that cannot be performed in a HLL, e.g., in many processor architectures, there are registers that are not accessible in C.

### Java ME<sup>58</sup>

If people claim they are using Java in an ES, then as far as I am concerned, it's not really an ES. Sure, Android apps are written in Java, but since a smart phone ain't an ES, that don't count.

### C#

See Java ME. I am clueless.

53 [http://en.wikipedia.org/wiki/Programmable\\_Interrupt\\_Controller](http://en.wikipedia.org/wiki/Programmable_Interrupt_Controller)

54 [http://en.wikipedia.org/wiki/Analog-to-digital\\_converter](http://en.wikipedia.org/wiki/Analog-to-digital_converter)

55 [http://en.wikipedia.org/wiki/Digital-to-analog\\_converter](http://en.wikipedia.org/wiki/Digital-to-analog_converter)

56 [http://blog.vdcresearch.com/embedded\\_sw/2010/09/what-languages-do-you-use-to-develop-software.html](http://blog.vdcresearch.com/embedded_sw/2010/09/what-languages-do-you-use-to-develop-software.html). Although the data is 5 years old, I doubt the distribution has changed much. ES programmer seem much less likely to jump on X-is-the-next-best-language-bandwagon the way web developers are because C is such a great language.

57 [http://en.wikipedia.org/wiki/Embedded\\_C%2B%2B](http://en.wikipedia.org/wiki/Embedded_C%2B%2B)

58 [http://en.wikipedia.org/wiki/Java\\_Platform,\\_Micro\\_Edition](http://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition)

**Ada**<sup>59</sup>

Ada is a statically-typed, imperative, and OO programming language designed under contract to the US Department of Defense in the early 1980's. It was specifically designed for embedded and real-time systems programming. It is based on the Pascal programming language, and if you've studied any VHDL<sup>60</sup> you will notice that VHDL syntax is similar in many respects to Ada. In the mid 1980's, Ada was widely perceived to be the next great programming language and the language to end all languages. This never came to pass, and today, Ada is mainly used only in military and aerospace applications.

**FORTH**<sup>61</sup>

I like FORTH, and it has a small, but devout, following in the ES community, but it's not very popular.

**4 System-on-Chip (SoC)<sup>62</sup> and System-in-Package (SiP)<sup>63</sup> Design**

SoC stands for either system-on-chip or system-on-a-chip. Regardless of which you prefer, the intent is the same: to squeeze pretty much an entire computer system onto a single chip. According to<sup>64</sup> the first SoC was constructed in 1974 to power a Microma digital watch<sup>65</sup>, but it was not until the last ten or so years that SoC's have become widely used, specifically in the smart phone and pad computer markets.

An SoC will contain one or microprocessors and/or microcontrollers, and some or all of these components,

- Other application-specific digital circuits.
- A digital signal processing (DSP) core (this is a form of ASIP).
- Memory (RAM, ROM, EEPROM, or flash).
- A graphics processing unit (GPU).
- Clocking sources (oscillators and PLL's).
- Various peripherals (e.g., timers, counters, real-time clocks, PWM).
- External I/O interfaces (e.g., Ethernet, UART, Firewire, USB, I<sup>2</sup>C, SPI, SATA, GPIO, HDMI, memory).
- A/D and D/A converters.
- Voltage regulators and power management circuits.
- Analog and mixed signal circuitry, including RF circuitry.
- Interconnect circuitry, i.e., to connect the various circuits and components.

SoC's can be implemented using full-custom VLSI, ASIC's, and FPGA's, and are primarily designed using reusable intellectual property (IP) blocks from various vendors, e.g., an ARM microprocessor, a PowerVR GPU, etc. The chip designer selects blocks that will permit the system to function as intended while meeting various design constraints, including cost, power-consumption, area, and interoperability (i.e., does this IP block play well with the other IP blocks in the system).

59 [http://en.wikipedia.org/wiki/Ada\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Ada_(programming_language))

60 VHSIC Hardware Description Language. <http://en.wikipedia.org/wiki/VHDL>

61 [http://en.wikipedia.org/wiki/Forth\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Forth_(programming_language))

62 [http://en.wikipedia.org/wiki/System\\_on\\_chip](http://en.wikipedia.org/wiki/System_on_chip)

63 [http://en.wikipedia.org/wiki/System\\_in\\_package](http://en.wikipedia.org/wiki/System_in_package)

64 <http://www.computerhistory.org/semiconductor/timeline/1974-digital-watch-is-first-system-on-chip-integrated-circuit-52.html>

65 I am old enough to remember how cool it was to get a digital watch for Christmas in 1976. I also distinctly remember getting my first 4-operation calculator around that same time (calculators began to be affordable around 1973-1974). I wish I had kept those things. Incidentally, Intel purchased Microma and very quickly got out of the digital watch business as prices began to plummet.





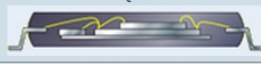
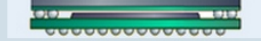
**Example SoC: Apple A8<sup>66</sup>**

The Apple A8 first appeared in the iPhone 6 and iPhone 6 Plus smart phones in 2014. Apple likes to keep its technology a secret, but these features are known to be present,

- Dual-core, 1.38 GHz, 64-bit ARMv8-A core.
- PowerVR 6XT GX6450 quad-core GPU (hardware accelerator for audio and video applications).
- 1 GiB of low power DDR3 dynamic RAM.
- 64 KiB instruction and 64 KiB data L1 cache. 1 MiB shared L2 cache. 4 MiB L3 cache.
- Feature size 20 nm.
- Two billion transistors.



SiP stands for either system-in-package or system-in-a-package<sup>67</sup>. Regardless of which you prefer, the idea is similar to SoC, but rather than cramming the entire system on one chip, the system is composed of two or more chips which are arranged to form a package. The following image illustrates different SiP package technologies used by Renesas<sup>68</sup>.

Structure	Advantage	Flexibility of Memory vender	High Density	Heat Dissipation	Cost
 <p>Chip Stack</p>	Small and High-density Circuits	☆	☆☆☆	☆☆	☆☆
 <p>Side by Side</p>	Good heat dissipation	☆	☆	☆☆☆	☆
 <p>QFP</p>	Low Cost	☆	☆	☆☆	☆☆☆
 <p>PoP (Package on Package)</p>	High Flexibility of Memory vender	☆☆☆	☆☆	☆☆	☆

The fourth picture (above) shows the package-on-package<sup>69</sup> technology in which two packages are stacked one on top of the other. Commonly PoP will be used to stack two memory packages or stack an SoC on the bottom (which is in a package) with a memory package on the top.

<sup>66</sup> [https://en.wikipedia.org/wiki/Apple\\_A8](https://en.wikipedia.org/wiki/Apple_A8)

<sup>67</sup> [https://en.wikipedia.org/wiki/System\\_in\\_package](https://en.wikipedia.org/wiki/System_in_package)

<sup>68</sup> <http://am.renesas.com/products/package/trend/sip/index.jsp>

<sup>69</sup> [http://en.wikipedia.org/wiki/Package\\_on\\_package](http://en.wikipedia.org/wiki/Package_on_package)

## References

[ARM-ARM]	<i>ARM Architecture Reference Manual</i> , Rev I., ARM, 2005.
[ARM-CALL]	<i>Procedure Call Standard for the ARM Architecture</i> , Rev. E 2.09, ARM, 2012.
[ARM-ISQR]	<i>ARM and Thumb-2 Instruction Set Quick Reference Card</i> , Rev. M, ARM, 2008.
[ARM-M0+DUG]	<i>Cortex-M0+ Device User's Guide</i> , ARM, 2012.
[ARM-M0+TRM]	<i>Cortex-M0+ Technical Reference Manual</i> , Rev. r0p1, ARM, 2012.
[ARM-V6M]	<i>ARMv6-M Architecture Reference Manual</i> , Rev. C, 2010.
[BARR]	Barr, M., <i>Programming Embedded Systems in C and C++</i> , O'Reilly, 1999.
[BERGER]	Berger, A, <i>Embedded Systems Design</i> , CMP Books, 2002.
[FRDM-KL46-PIN]	<i>FRDM-KL46Z Pinout</i> .
[FRDM-KL46Z-QSG]	<i>FRDM-KL46Z Quick Start Guide</i> , Rev. 1, Freescale, 2013.
[FRDM-KL46-SCH]	<i>FRDM-KL46Z Schematic</i> , Rev. C, Freescale, 2013.
[FRDM-KL46Z-UM]	<i>FRDM-KL46Z User's Manual</i> , Rev. 1.0, Freescale.
[HEATH]	Heath, S., <i>Embedded Systems Design</i> , 2nd ed, Newnes, 2003.
[KL46-SDS]	<i>Kinetis KL46 Sub-Family Data Sheet</i> , Rev. 5, Freescale, 2014.
[KL46-SRM]	<i>Kinetis KL46 Sub-Family Reference Manual</i> , Rev. 3, Freescale, 2013.
[NOER]	Noergaard, T., <i>Embedded Systems Architecture</i> , 1st ed, Newnes, 2005.
[OPENSDA]	<i>OpenSDA User's Guide</i> , Rev. 0.93, Freescale, 2012.
[VAHID]	Vahid, F., and Givargis, T., <i>Embedded System Design</i> , John Wiley & Sons, 2002.
[WHITE]	White, E., <i>Making Embedded Systems</i> , O'Reilly, 2011.