Embedded Systems

4th Stage

Lecture Four

Assist. Prof. Dr. Yasir Amer Abbas

Computer Engineering Department

2022

Embedded System Hardware

Lecture Four

Inputs for Embedded System Hardware Processing Unit

4. Embedded System Hardware

4.1 Introduction

It is one of the characteristics of embedded and cyber-physical systems that both hardware and software must be taken into account. The achievable performance depends crucially on the available hardware platform. Moreover, motivation for this was already included in the Preface: "*The development of embedded systems cannot ignore the underlying hardware characteristics. Timing, memory usage, power consumption, and physical failures are important*".

4. Embedded System Hardware

4.1 Introduction

The reuse of available hard- and software components is at the heart of the platform-based design methodology. Consistent with the need to consider available hardware components and with the design information flow shown in Fig. 1, we are now going to describe some of the essentials of embedded system hardware.

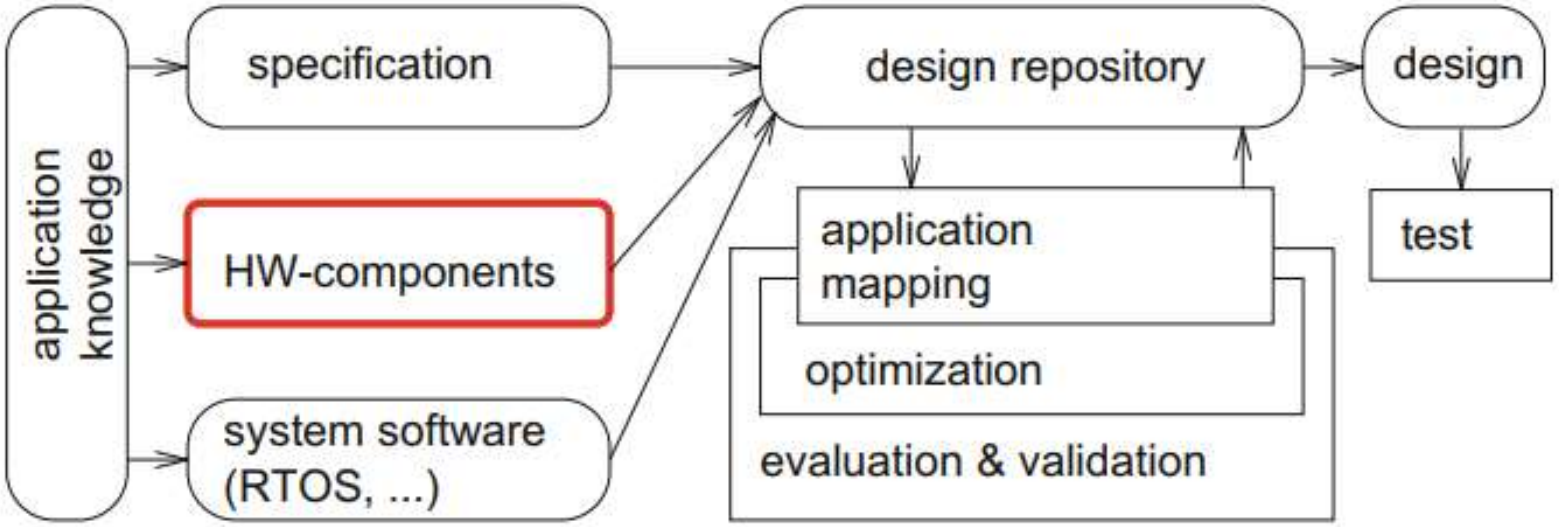


Fig. 1 Simplified design information flow

4. Embedded System Hardware

4.1 Introduction

Hardware for embedded systems is much less standardized than hardware for personal computers. Due to the huge variety of embedded system hardware, it is impossible to provide a comprehensive overview of all types of hardware components. Nevertheless, we will try to provide a survey of some of the essential components which can be found in most systems

In many cyber-physical systems, especially in control systems, hardware is used in a loop shown in the **Fig. 2.** We will use this loop to structure the presentation of components in this lectuer

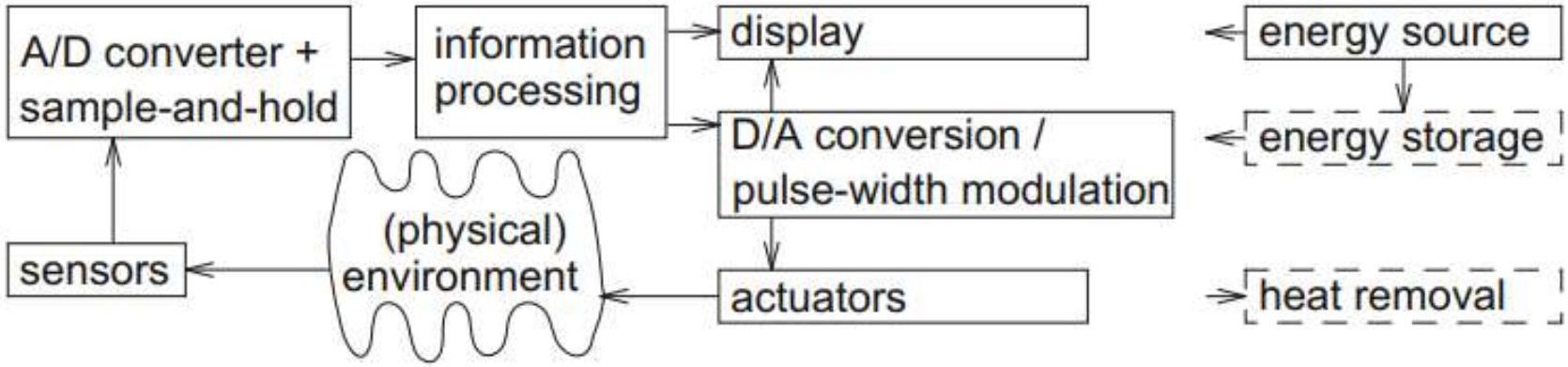


Fig. 2 Hardware in the loop

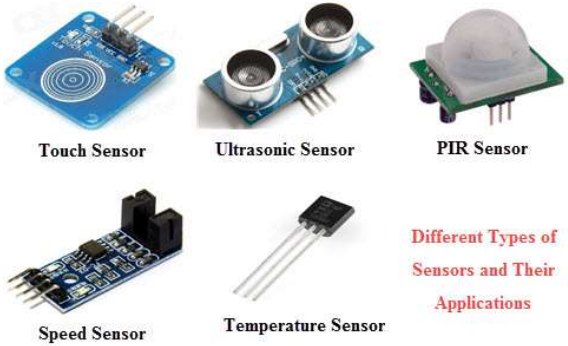
4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors

There are sensors for weight, velocity, acceleration, electrical current, voltage, temperature, etc. A wide variety of physical effects can be exploited in the construction of sensors. Examples include the law of induction (generation of voltages in an electric field), and photoelectric effects. There are also sensors for chemical substances.

Recent years have seen the design of a huge range of sensors, and much of the progress in designing smart systems can be attributed to modern sensor technology. The availability of sensors has enabled the design of sensor networks, a key element of the Internet of Things. It is impossible to cover this subset of cyber-physical hardware technology comprehensively, and we can only give characteristic examples:



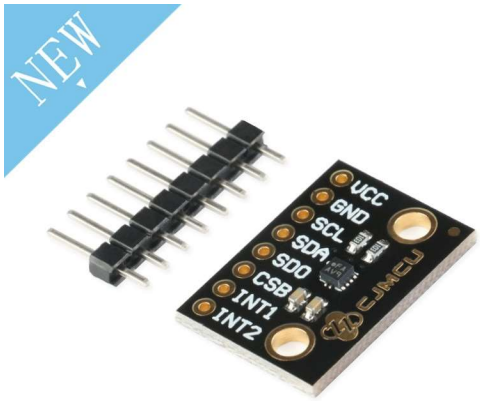
4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors (example of Sensors)

Acceleration sensors: A small sensor manufactured using microsystem technology. The sensor contains a small mass in its center. When accelerated, the mass will be displaced from its standard position, thereby changing the resistance of the tiny wires connected to the mass.

Acceleration sensors are included in the powerful inertial measurement units (IMUs). They contain gyros and accelerometers, and they capture up to six degrees of freedom, comprising position (*x, y*, and *z*) and orientation (roll, pitch, and yaw)



BMA400

4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors (example of Sensors)

Image sensors: There are essentially two kinds of image sensors: charge-coupled devices (CCDs) and CMOS (Complementary Metal Oxide Semiconductor) sensors. In both cases, arrays of light sensors are used. The architecture of CMOS sensor arrays is similar to that of standard memories: Individual pixels can be randomly addressed and read out. CMOS sensors use standard CMOS technology for integrated circuits. Due to this, sensors and logic circuits can be integrated on the same chip. This allows some preprocessing to be done already on the sensor chip, leading to so-called smart sensors. CMOS sensors require only a single standard supply voltage, and interfacing in general is easy. Therefore, CMOS-based sensors can be cheap.

4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors (example of Sensors)

Biometric sensors: Demands for higher security standards as well as the need to protect mobile and removable equipment have led to an increased interest in authentication. Due to the limitations of password-based security (e.g., stolen and lost passwords), biometric sensors and biomedical authentication receive attention. Biometric authentication tries to identify whether or not a certain person is actually the person she or he claims to be. Methods for biometric authentication include iris scans, finger print sensors, and face recognition. Finger print sensors are typically fabricated using the same CMOS technology which is used for manufacturing integrated circuits. CCD and CMOS image sensors can be used for face recognition. False accepts as well as false rejects are an inherent problem of biometric authentication. In contrast to password-based authentication, exact matches are not possible.

4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors (example of Sensors)

Artificial eyes: Artificial eye projects have received significant attention. Some projects have an impact on the eye, but others provide vision in an indirect way. For example, the Dobelle Institute experimented with a camera attached to a computer sending electrical pulses to a direct brain contact. More recently, the less invasive translation of images into audio has been preferred.

4. Embedded System Hardware

4.2 Inputs for Embedded System Hardware

4.2.1 Sensors (example of Sensors)

Radio frequency identification (RFID): RFID technology is based on the response of a **tag** to radio frequency signals. The tag consists of an integrated circuit and an antenna, and it provides its identification to **RFID readers**. The maximum distance between tags and readers depends on the type of the tag. The technology is used to identify objects, animals, or people and is a key enabler for the Internet of Things.

4. Embedded System Hardware

4.2.2 Discretization of Time: Sample-and-Hold Circuits

All known digital computers work in a **discrete** time domain *DT*. This means that they can process discrete sequences or **streams** of values. Hence, incoming signals over the continuous time domain must be converted to signals over the discrete time domain. This is the purpose of **sample-and-hold circuits**. Figure 3 (**left**) shows a simple sample-and-hold circuit. In essence, the circuit consists of a clocked transistor and a capacitor. The transistor operates like a switch. Each time the switch is closed by the clock signal, and the capacitor is charged so that its voltage *h(t)* is practically the same as the incoming voltage *e(t)*. After opening the switch again, this voltage will remain essentially unchanged until the switch is closed again. Each of the values stored on the capacitor can be considered as an element of a discrete sequence of values *h(t)*, generated from a continuous function *e(t)* (see Fig. 3 (**right**)). If we sample *e(t)* at times {*ts*}, then *h(t)* will be defined only at those times

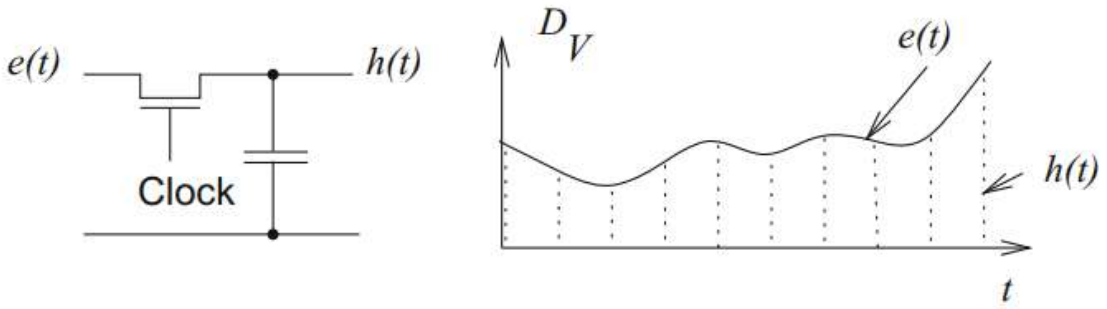


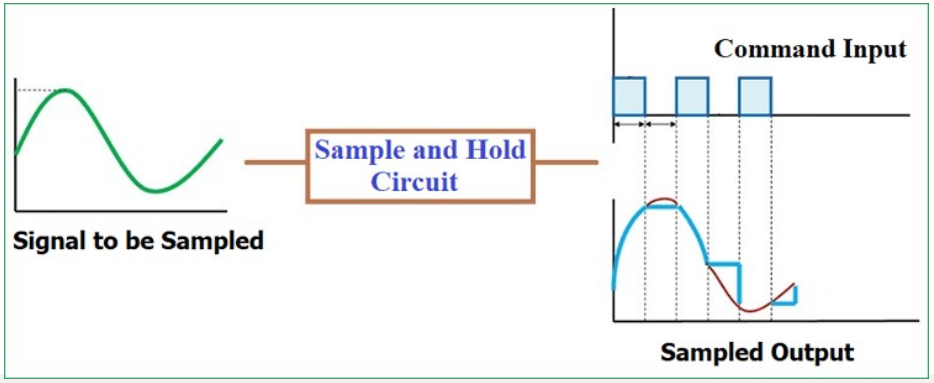
Fig. 3 Sample-and-hold phase: **left**: circuit; **right**: signals

4. Embedded System Hardware

4.2.2 Discretization of Time: Sample-and-Hold Circuits

Takes samples from the Analog input signal and hold them for particular period of time and then outputs the sampled part of input signal. This circuit is only useful for sampling few microseconds of input signal.

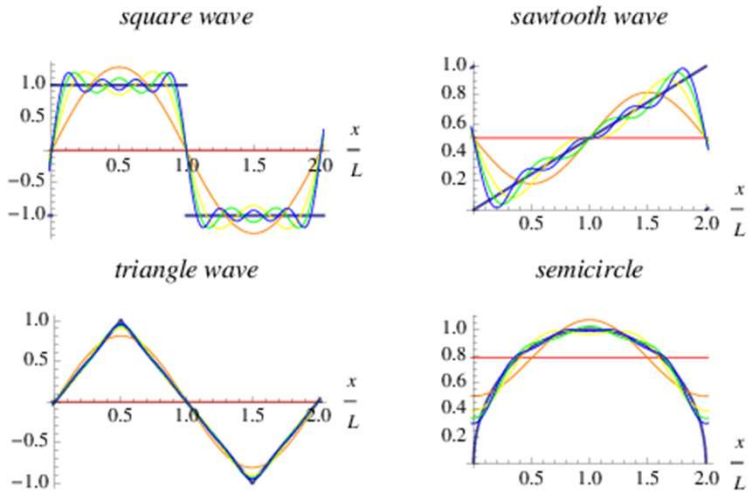
An ideal sample-and-hold circuit would be able to change the voltage at the capacitor in an arbitrarily short amount of time. This way, the input voltage at an particular instance in time could be transferred to the capacitor, and each element in the discrete sequence would correspond to the input voltage at a particular point in time. In practice, however, the transistor has to be kept closed for a short time window in order to really charge or discharge the capacitor. The voltage stored on the capacitor will then correspond to a voltage reflecting that short time window.



4. Embedded System Hardware

4.2.3 Fourier Approximation of Signals

The Fourier Series, the founding principle behind the field of Fourier Analysis, is an infinite expansion of a function in terms of sines and cosines. In physics and engineering, expanding functions in terms of sines and cosines is useful because it allows one to more easily manipulate functions that are, for example, discontinuous or simply difficult to represent analytically. In particular, the fields of electronics, quantum mechanics, and electrodynamics all make heavy use of the Fourier Series. Additionally, other methods based on the Fourier Series, such as the FFT (Fast Fourier Transform – a form of a Discrete Fourier Transform [DFT]), are particularly useful for the fields of Digital Signal Processing (DSP) and Spectral Analysis.



4. Embedded System Hardware

4.2.4 Discretization of Values: Analog-to-Digital Converters

The main purpose of the A/D converters within a data acquisition system is to convert conditioned analog signals into a stream of digital data so that the data acquisition system can process them for display, storage, and analysis. There is a large range of ADCs with varying speed/precision characteristics. Typically, fast ADCs have a low precision and high precision converters are slow. several converters flow:

4.2.4.1 Flash ADC

This type of ADCs uses a large number of comparators. Each comparator has two inputs, denoted as + and -. If the voltage at input + exceeds that at input -, the output corresponds to a logical ’1’ and it corresponds to a logical ’0’ otherwise

4.2.4.2 Successive Approximation

Distinguishing between a large number of digital values is possible with ADCs using successive approximation.

4.2.4.3 Pipelined Converters

These converters consist of a chain of converters, where each stage in the chain is in charge of converting a few bits. Each stage passes the remaining residue of the voltage to the next stage. For example, each stage could convert a single bit and subtract the corresponding voltage. The resulting residue would typically be scaled up by a factor of two (in order to avoid too small voltages) and be passed on to the next stage.

4. Embedded System Hardware

4.2.4 Discretization of Values: Analog-to-Digital Converters

4.2.4.4 Quantization Noise.

4.2.4.5 Integrating converters.

4.2.4.6 Folding ADCs.

4.2.4.7 Delta-Sigma ADCs

Comparison of ADCs

Figure 4 provides an overview of the speed/resolution trade-offs of ADCs, using a trade-off analysis of Vogels. Flash ADCs are clearly the fastest, but provide only a small resolution. Pipelining is frequently superior to successive approximation.

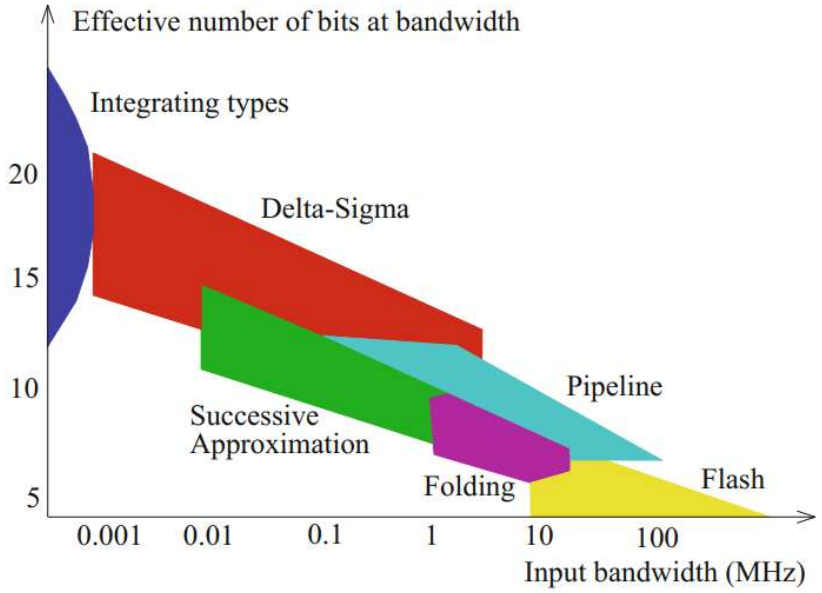


Fig. 4 Comparison of the speed/resolution characteristics of various ADCs

4. Embedded System Hardware

4.3 Processing Unit

The processing in embedded systems can consider to ASICs (application-specific integrated circuits) using hardwired multiplexed designs, reconfigurable logic, and several types of programmable processors.

4.3.1 Application-Specific Circuits (ASICs)

For high-performance applications and for large markets, application-specific integrated circuits (ASICs) can be designed. In general, ASICs are very energy-efficient . However, the cost of designing and manufacturing such chips is quite high. The cost of the mask set (which is used for transferring geometrical patterns onto the chip). However, it is feasible to decrease this cost by using less advanced semiconductor fabrication technologies and by using multi-project wafers (MPW) containing several designs. But there is a lack of flexibility: Correcting design errors typically requires a new mask set and a new fabrication run (unless the ASIC contains processors with writable memories). This approach also has to cope with potentially large design efforts requiring dedicated skills and expensive tools. Therefore, ASICs are appropriate only under special circumstances, such as large market volumes, ultimate energy efficiency demands, special voltage or temperature ranges, mixed analog/digital signals, or security-driven designs.

4. Embedded System Hardware

4.3 Processing Unit

4.3.2 Processors

The key advantage of processors is their **flexibility**. With processors, the overall behaviour of embedded systems can be changed by just changing the software running on those processors. Changes of the behaviour may be required in order to correct design errors, to update the system to a new or changed standard, or to add features to the previous system. Because of this, processors have found widespread use in embedded systems. In particular, processors which are available commercially "off the-shelf" (COTS) have become very popular. Embedded processors must be used in a resource-aware manner, i.e., we need to care about resources required for running applications on them. Furthermore, they do not need to be instruction set compatible with commonly used personal computers (PCs) or servers. Therefore, their architectures may be different from those processors. Efficiency has a number of different aspects which are discussed next.

4. Embedded System Hardware

4.3 Processing Unit

4.3.2 Processors

4.3.2.1 Energy Efficiency

Minimization of power and energy consumption are both important. Power consumption has an effect on the size of the power supply, the design of the voltage regulators, the dimensioning of the interconnect, and short-term cooling. Minimizing the energy consumption is required especially for mobile applications, since battery technology is only slowly improving, and since the cost of energy may be quite high. Also, a reduced energy consumption decreases cooling requirements and improves the reliability (since the lifetime of electronic circuits decreases for high temperatures. The **dynamic power consumption** is the power consumption caused by switching (in contrast to the **static power consumption** which exists even if no switching takes place) At least three techniques can be applied at a rather high level of abstraction

• **Parallel execution**: Parallel execution is an effective means of improving the overall energy efficiency.

• **Dynamic power management (DPM)**: With this approach, processors have several power-saving states in addition to the standard operating state. Each power saving state has a different power consumption and a different time for transitions into the operating state.

. **Dynamic voltage and frequency scaling (DVFS)**: For example, the Crusoe™processor by Transmeta provided 32 voltage levels between 1.1and 1.6Volts, and the clock could be varied between 200MHz and 700MHz in increments of 33MHz.

4. Embedded System Hardware

4.3 Processing Unit

4.3.2 Processors

4.3.2.2 Code Size Efficiency

Minimizing the code size is very important for embedded systems, since large hard disk drives(HDDs) or solid-state disks(SSDs) are typically not available and since the capacity of memory is typically also very limited. This is even more pronounced for **systems on a chip** (SoCs). For SoCs, the memory and processors are implemented on the same chip. In this particular case, memory is called **embedded memory**.

Embedded memory may be more expensive to fabricate than separate memory chips, since the fabrication processes for memories and processors must be compatible. Nevertheless, a large percentage of the total chip area may be consumed by the memory. There are several techniques for improving the code size efficiency:

• **CISC machines**: Standard RISC processors have been designed for speed, not for code size efficiency. Earlier complex instruction set processors (CISC machines) were actually designed for code size efficiency, since they had to be connected to slow memories. Caches were not frequently used. Therefore, "old-fashioned" CISC processors are finding applications in embedded systems. Cold Fire processors, which are based on the Motorola 68000 family of CISC processors, are an example.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.2 Code Size Efficiency

• **Compression techniques**: In order to reduce the amount of silicon needed for storing instructions as well as in order to reduce the energy needed for fetching these instructions, instructions are frequently stored in memory in compressed form. This reduces both the area as well as the energy necessary for fetching instructions. Due to the reduced bandwidth requirements, fetching can also be faster. The goals of compression can be summarized as follows:

1. Save ROM and RAM areas, since these may be more expensive than the processors themselves.

2. Use some encoding technique for instructions and possibly also for data with the following properties:

A. There should be little or no run-time penalty for these techniques.

B. Decoding should work from a limited context (it is, for example, impossible to read the entire program to find the destination of a branch instruction).

C. Word sizes of the memory, of instructions, and of addresses must be taken into account.

D. Branch instructions branching to arbitrary addresses must be supported.

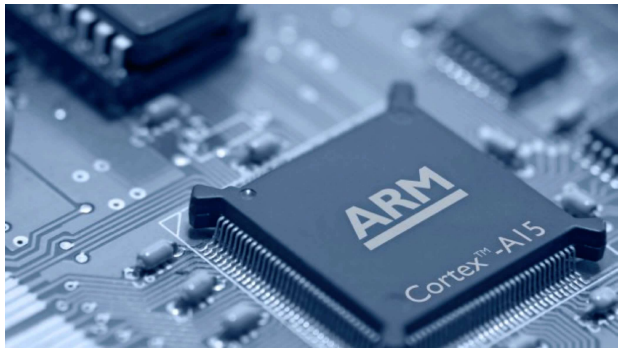
E. Fast encoding is only required if writable data is encoded. Otherwise, fast decoding is sufficient.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.2 Code Size Efficiency

For some processors, there is a **second instruction set**. This second instruction set has a narrower instruction format. An example of this is the ARM® processor family. The original ARM **Advanced RISC Machine** instruction set is a 32-bit instruction set and includes **predicated execution**. This means an instruction is executed if and only if a certain condition concerning values in the condition code registers is met. This condition is encoded in the first four bits of the instruction format. Most ARM processors also provide a second instruction set, with 16-bit wide instructions, called THUMB instructions. THUMB instructions are shorter, since they do not support predication, use shorter and less register fields, and use shorter immediate fields. THUMB instructions are dynamically converted into ARM instructions while programs are decoded.



4. Embedded System Hardware

4.3.2 Processors

4.3.2.4 Digital Signal Processing (DSP)

In addition to allowing single instruction realizations of loop bodies for filtering, DSP processors provide a number of other application domain-oriented features:

• **Specialized addressing modes**: In the filter application described above, only the last *n* elements of *w* need to be available. Ring buffers can be used for that. These can be implemented easily with modulo addressing**. In modulo addressing, addresses can be incremented and decremented until the first or last element of the buffer is reached**. Additional increments or decrements will result in addresses pointing to the other end of the buffer.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.4 Digital Signal Processing (DSP)

• **Separate address generation units**: Address generation units (AGUs) are typically directly connected to the address input of the data memory (see Fig. 5). **Addresses which are available in address registers can be used in register-indirect addressing modes**. **This saves machine instructions, cycles, and energy.** In order to increase the usefulness of address registers, instruction sets typically contain auto-increment and auto-decrement options for most instructions using address registers.

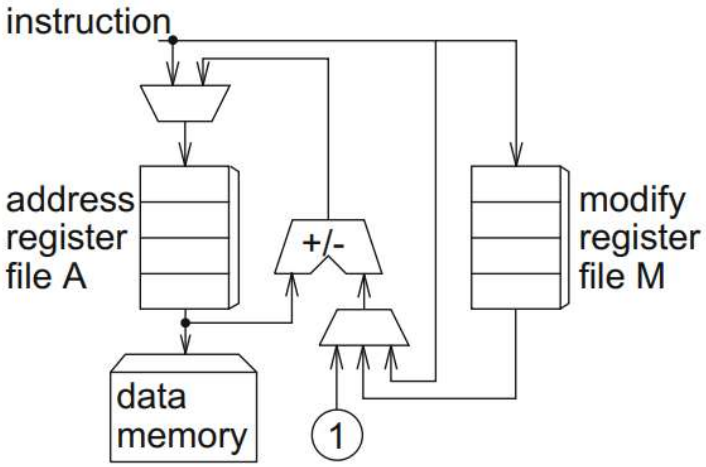


Fig 5 AGU using special address registers

Question

- What is Overflow?

an **overflow** error can occur when a calculation is run but the **computer** is unable to store the answer correctly

4. Embedded System Hardware

4.3.2 Processors

4.3.2.4 Digital Signal Processing (DSP)

• **Saturating arithmetic**: Saturating arithmetic changes the way overflows and underflows are handled. In standard binary arithmetic, wraparound is used for the values returned after an overflow or underflow.

Fixed-point arithmetic: Floating-point hardware increases the cost and power consumption of processors. Consequently, it has been estimated that 80% of the DSP processors do not include floating-point hardware.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.5 Microcontrollers

A large number of the processors in embedded systems are in fact microcontrollers. Microcontrollers are typically not very complex and can be used easily. Due to their relevance for designing control systems, One of the most frequently used processors is Intel 8051, available from many vendors (but no longer from Intel). This processor has the following characteristics:

- 8-bit CPU, optimized for control applications,
- large set of operations on Boolean data types,
- program address space of 64 k bytes,
- separate data address space of 64 k bytes,
- 4 k bytes of program memory on chip, 128 bytes of data memory on chip,
- 32 I/O lines, each of which can be addressed individually,
- 2 counters on the chip,
- universal asynchronous receiver/transmitter for serial lines available on the chip,
- clock generation on the chip,
- many variations commercially available.

All these characteristics are quite typical for microcontrollers.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.6 Multimedia Processors/Instruction Sets

Registers and arithmetic units of many modern architectures are at least 64 bits wide. Therefore, two 32-bit data types ("double words"), four 16-bit data types ("words"), or eight 8-bit data types ("bytes") can be packed into a single register (see Fig. 6).

Fig. 6 Using 64-bit registers for packed words

Arithmetic units can be designed such that they suppress carry bits at double word, word, or byte boundaries. Multimedia instruction sets exploit this fact by supporting operations on packed data types. Such instructions are sometimes called single instruction, multiple-data (SIMD) instructions, since a single instruction encodes operations on several data elements. With bytes packed into 64-bit registers, speedups of up to about eight over non-packed data types are possible. Data types are typically stored in packed form in memory. Unpacking and packing are avoided if arithmetic operations on packed data types are used.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.6 Multimedia Processors/Instruction Sets

Furthermore, multimedia instructions can usually be combined with saturating arithmetic and, therefor, provide a more efficient form of overflow handling than standard instructions. Hence, the overall speedup achieved with multimedia instructions can be significantly larger than the factor of eight enabled by operations on packed data types. Due to the advantages of operations on packed data types, new instructions have been added to several processors.

For example, so-called **streaming SIMD extensions** (SSE) have been added to Intel's family of Pentium®-compatible processors. New instructions have also been called **short vector instructions** and introduced by Intel® as advanced vector extensions (AVX).

https://en.wikichip.org/wiki/intel/pentium

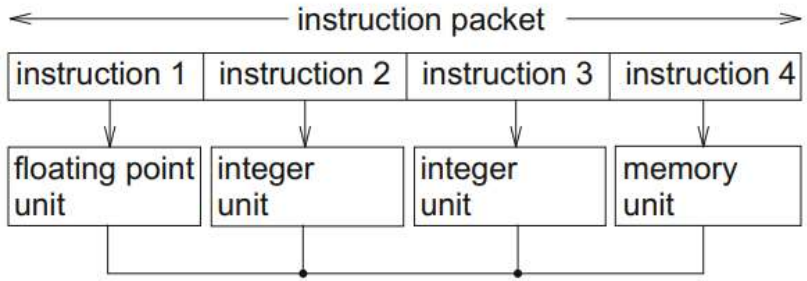
4. Embedded System Hardware

4.3.2 Processors

4.3.2.7 Very Long Instruction Word (VLIW) Processors

Computational demands for embedded systems are increasing, especially when multimedia applications, advanced coding techniques, or cryptography are involved. Performance improvement techniques used in high-performance microprocessors are not appropriate for embedded systems: Driven by the need for instruction set compatibility, processors found, for example, in PCs spend a huge amount of resources and energy on automatically finding parallelism in application programs. Still, their performance is frequently not sufficient. For embedded systems, we can exploit the fact that instruction set compatibility with PCs is not required. Therefore, we can use instructions which explicitly identify operations to be performed in parallel. This is possible with **explicit parallelism instruction set computers** (EPICs). With EPICs, ***detection of parallelism is moved from the processor to the compiler***. This avoids spending silicon and energy on the detection of parallelism at run-time. As a special case, its consider very long instruction word (VLIW) processors.

For VLIW processors, several operations or instructions are encoded in a long instruction word (sometimes called **instruction packet**) and are assumed to be executed in parallel. Each operation/instruction is encoded in a separate field of the instruction packet. Each field controls certain hardware units. Four such fields are used in Fig. 7, each one controlling one of the hardware units.



4. Embedded System Hardware

4.3.2 Processors

4.3.2.7 Very Long Instruction Word (VLIW) Processors

Partitioned Register Files

Implementing register files for **VLIW and EPIC** processors is far from trivial. ***Due to the large number of operations that can be performed in parallel, a large number of register accesses has to be provided in parallel***. Therefore, a large number of ports is required. However, the delay, size, and energy consumption of register files increase with their number of ports. Hence, register files with very large numbers of ports are inefficient. As a consequence, many VLIW/EPIC architectures use partitioned register files. Functional units are then only connected to a subset of the register files. As an example, Fig.8 . shows the internal structure of the TMS 320C6xx processors. These processors have two register files, and each of them is connected to half of the functional units. During each clock cycle, only a single path from one register file to the functional units connected to the other register file is available.

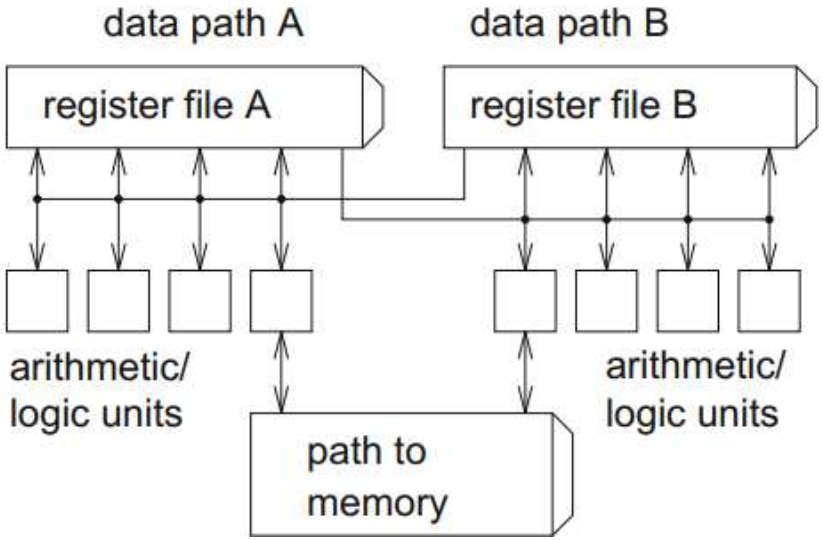


Fig 8 Partitioned register files for TMS 320C6xx

4. Embedded System Hardware

4.3.2 Processors

4.3.2.7 Very Long Instruction Word (VLIW) Processors

Predicated Execution

A potential problem of VLIW and EPIC architectures is their possibly large **delay penalty**: This delay penalty might originate from branch instructions found in some instruction packets. Instruction packets normally must pass through pipelines. Each stage of these pipelines implements only part of the operations to be performed by the instructions executed. The fact that branch instructions exist cannot be detected in the first stage of the pipeline. When the execution of the branch instruction is finally.

There are essentially two ways to deal with these additional instructions:

1. They are executed as if no branch had been present. This case is called **delayed branch**. Instruction packet slots that are still executed after a branch are called **branch delay slots**. These branch delay slots can be filled with instructions which would be executed before the branch if there were no delay slots. However, it is normally difficult to fill all delay slots with useful instructions and some must be filled with no-operation instructions (NOPs). The term **branch delay penalty** denotes the loss of performance resulting from these NOPs.

2. The pipeline is stalled until instructions from the branch target address have been fetched. There are no branch delay slots in this case. In this organization, the branch delay penalty is caused by the stall .

4. Embedded System Hardware

4.3.2 Processors

4.3.2.7 Very Long Instruction Word (VLIW) Processors

predicated instructions: Predication can be used to implement small if-statements efficiently: The condition is stored in one of the condition registers and if-statement bodies are implemented as predicated instructions which depend on this condition. This way, if-statement bodies can be evaluated in parallel with other operations and no delay penalty is incurred. The Crusoe™processor is a (commercially finally unsuccessful) example of an EPIC processor designed for PCs . Its instruction set includes 64-bit and 128-bit VLIW instructions. Efforts for making EPIC instruction sets available in the PC sector resulted in Intel's IA-64 instruction set and its implementation in the Itanium® processor. Due to legacy problems, it has been used mainly in the server market. Many MPSoCs are based on VLIW and EPIC processors.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.8 Multi-core Processors

Processor features for single processors described above have helped to design high performance processors in a resource-aware manner. However, it turned out that a further performance increase for single processors hits the **power wall**: a further increase in clock speeds would result in a too large power consumption and in too hot circuits. Further increase in the level of VLIW parallelism was not feasible either. Due to advances in fabrication technology, it is now feasible to manufacture multiple processors on the same semiconductor die. Multiple processors integrated on the same chip are called **multi-cores**. This is in contrast to multiprocessor systems which have been used in computing centers for decades. The integration of multiple cores on the same die enables a much faster communication, compared to multiprocessor systems. Also, this approach facilitates the sharing of resources (such as caches) among the cores. As an example, Fig. 9. demonstrates the architecture of the Intel® Core™ Duo.

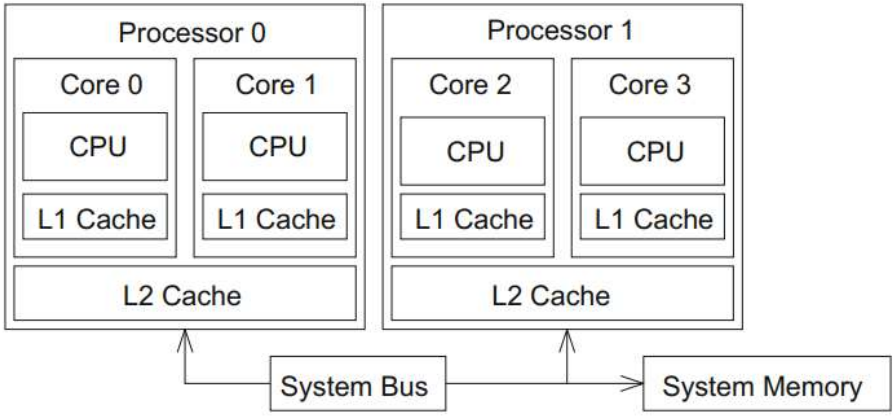


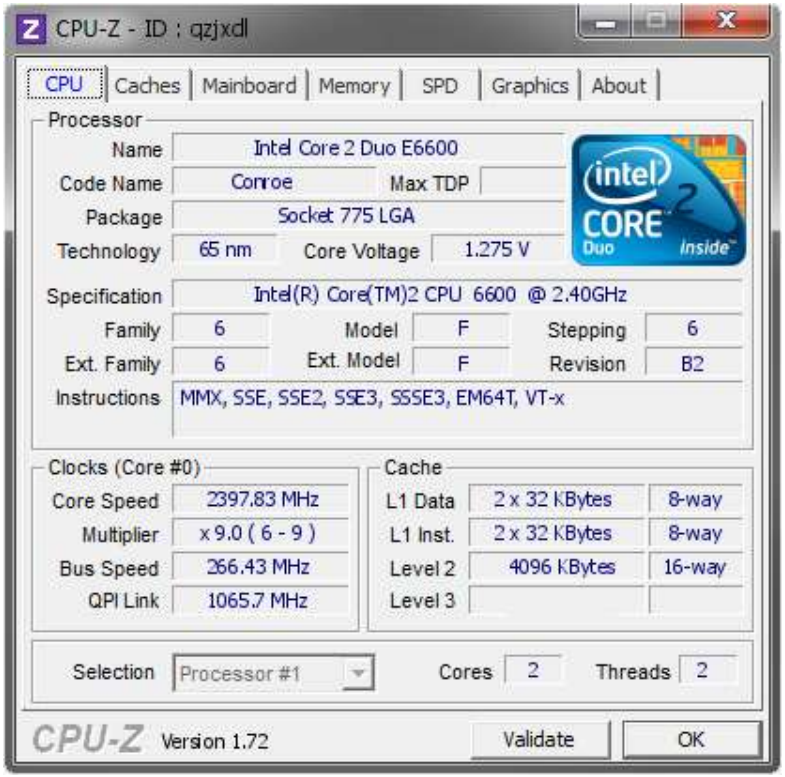
Fig. 9
Intel® Core™ Duo.

4. Embedded System Hardware

4.3.2 Processors

4.3.2..8 Multi-core Processors

In **Multi-core Processors** case, L1 caches are private, whereas L2 are shared. Implementing efficient accesses to caches needs some consideration. With such architectures, cache coherence is becoming an issue also within one die. This means, we have to know whether updates of data and possibly also instructions by one core are seen by the others. Protocols for automatic cache coherence, such as the MESI protocol, are known for many years in computer architecture. Now, they have to be implemented on the chip. Scalability is an issue: for how many cores can we reasonably provide enough bandwidth in the communication architecture to always keep caches coherent? Also, the system memory bandwidth may be insufficient for a growing number of cores. Architectures other than the above Intel architecture exist.



4. Embedded System Hardware

4.3.2 Processors

4.3.2..8 Multi-core Processors

In the architecture of Fig. 9. , all processors are of the same type. Such an architecture is called a **homogeneous multi-core** architecture. Advantages of homogeneous multi-core architectures include the fact that the design effort is limited (processors will be replicated) and that software can easily be migrated from one processor to another one. This is very useful in case one of the cores fails. In contrast to homogeneousmulti-core architectures, there are also **heterogeneous multi-core architectures**. In this case, processors are of different types. In this way, we can select processors which are best suited for certain applications or run-time scenarios. Typically, heterogeneous architectures must be used to achieve the best energy efficiency that is feasible.

In order to find a good compromise between homogeneous and (totally) heterogeneous architectures, architectures with a single instruction set but different internal architectures, so-called **single-ISA heterogeneous multi-cores**, have been proposed. The ARM® big.LITTLE architecture is a very prominent example of this. Figure 10 contains the pipeline architecture of the Cortex® A-15 processor.

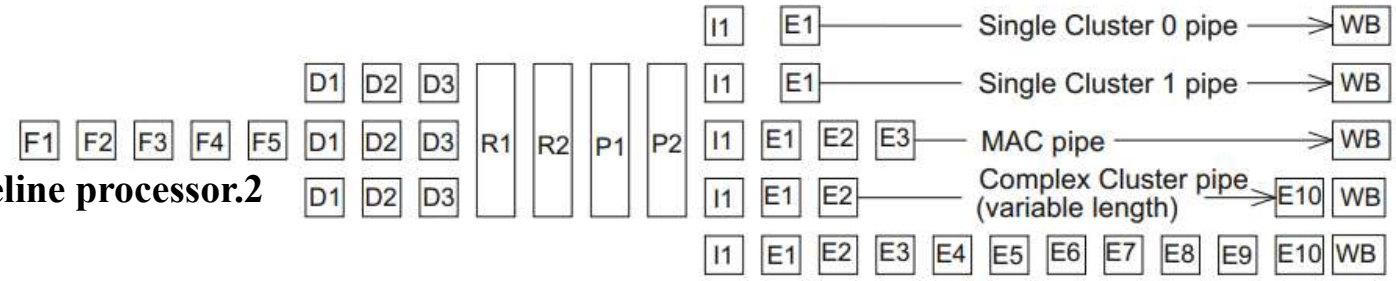


Fig. 10
ARM Cortex® A-15 Pipeline processor.2

4. Embedded System Hardware

4.3.2 Processors

4.3.2..8 Multi-core Processors

It is a relatively complex pipeline, containing multiple pipeline stages for instruction fetch, instruction decoding, instruction issue, execution and write-back. Using this architecture, instructions have to pass through at least 15 pipeline stages before their result is stored. Dynamic scheduling of instructions allows executing instructions in a sequence different from the one in which they are fetched from memory (so-called out-of-order execution). Several instructions can be issued in one clock cycle (so-called multi-issue). As a result, the architecture offers a high performance, but requires a substantial amount of power. In contrast, Fig. 11 shows the pipeline architecture of the Cortex® A-7 processor.

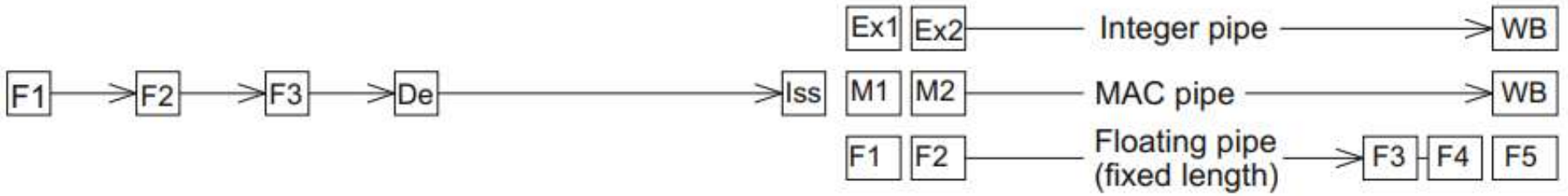


Fig. 11
ARM Cortex A-7 Pipeline.

4. Embedded System Hardware

4.3.2 Processors

4.3.2.3 Execution-Time Efficiency

In order to meet time constraints without having to use high clock frequencies, architectures can be customized to certain application domains, such as digital signal processing (DSP). One can even go one step further and design application-specific instruction set processors (ASIPs). As an example of domain-specific processors, we will consider processors for DSP. In digital signal processing, digital filtering is a very frequent operation. Overall, instructions pass through eight to eleven stages, they are always processed in the order in which they are fetched from memory and there is only a limited set of situations in which two instructions are issued at the same time. Due to this, the architecture requires very little power, but has a limited performance.

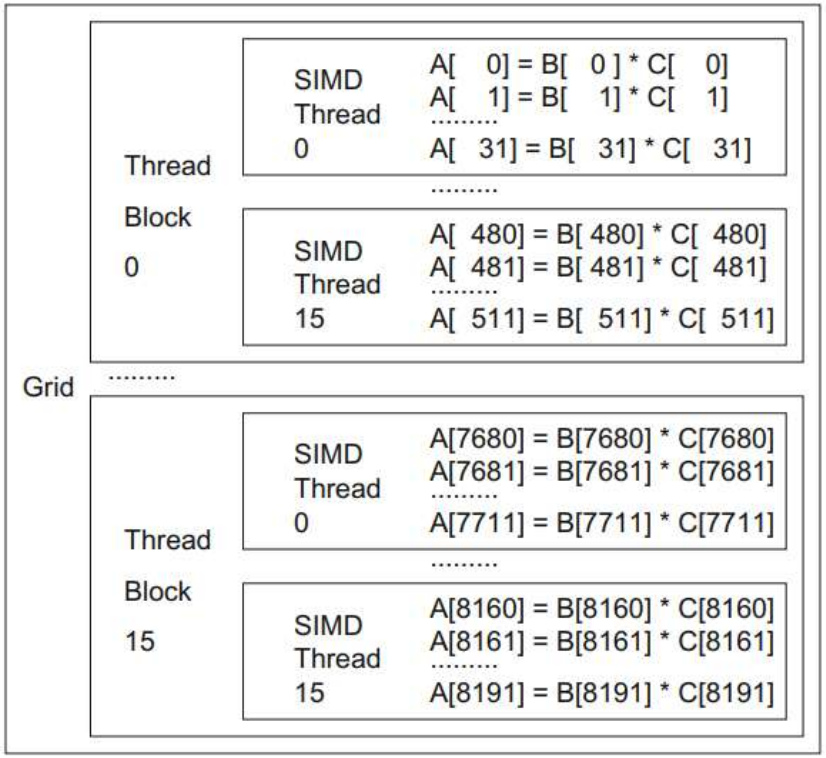
4. Embedded System Hardware

4.3.2 Processors

4.3.2.9 Graphics Processing Units (GPU)

In the last century, many computers used specialized graphics processing units (GPUs) in order to generate an appealing graphical representation of computer output. This hardwired solution suffered from being unable to support non-standard computer graphics algorithms. Therefore, these highly specialized GPUs have been replaced by programmable solutions. Current GPUs try to run a large number of computations concurrently in order to achieve the desired performance. The standard approach to concurrency is to run many fine-grained threads at the same time.

The goal is to keep many processing units busy. As an example, let us consider the multiplication of two large matrices on a GPU. Figure 12 shows how the computations can be mapped to a GPU.



4. Embedded System Hardware

4.3.2 Processors

4.3.2. 1-0. Multiprocessor Systems-on-a-Chip (MPSoCs)

Going one step further, heterogeneous multi-core systems have also been merged with GPUs. Figure 13 shows a contemporary heterogeneous multi-core system, also comprising a Mali GPU Mapping techniques for such processors are important, since examples demonstrate that a power efficiency close to that of ASICs can be achieved. For example, for IMEC's ADRES processor, an efficiency of 55 ∗ 109 operations per Watt (about 50% of the power efficiency of ASICs) has been predicted. However, the design effort for such architectures is larger than in the homogeneous case.

The architecture shown in Fig. 13 does not only contain processor cores. Rather, it comprises a number of additional system components, such as memory management units and interfaces for peripheral devices. Overall, the idea behind this integration is to avoid extra chips for such functionality. As a result, a whole system is integrated on one chip. Therefore, we are calling such an architecture a system-on-a-chip (SoC) or even a multiprocessor system-on-a-chip (MPSoC) architecture.

Question

- ***GPU is SIMD or MIMD?***

4. Embedded System Hardware

4.3.2 Processors

4.3.2. 1-0. Multiprocessor Systems-on-a-Chip (MPSoCs)

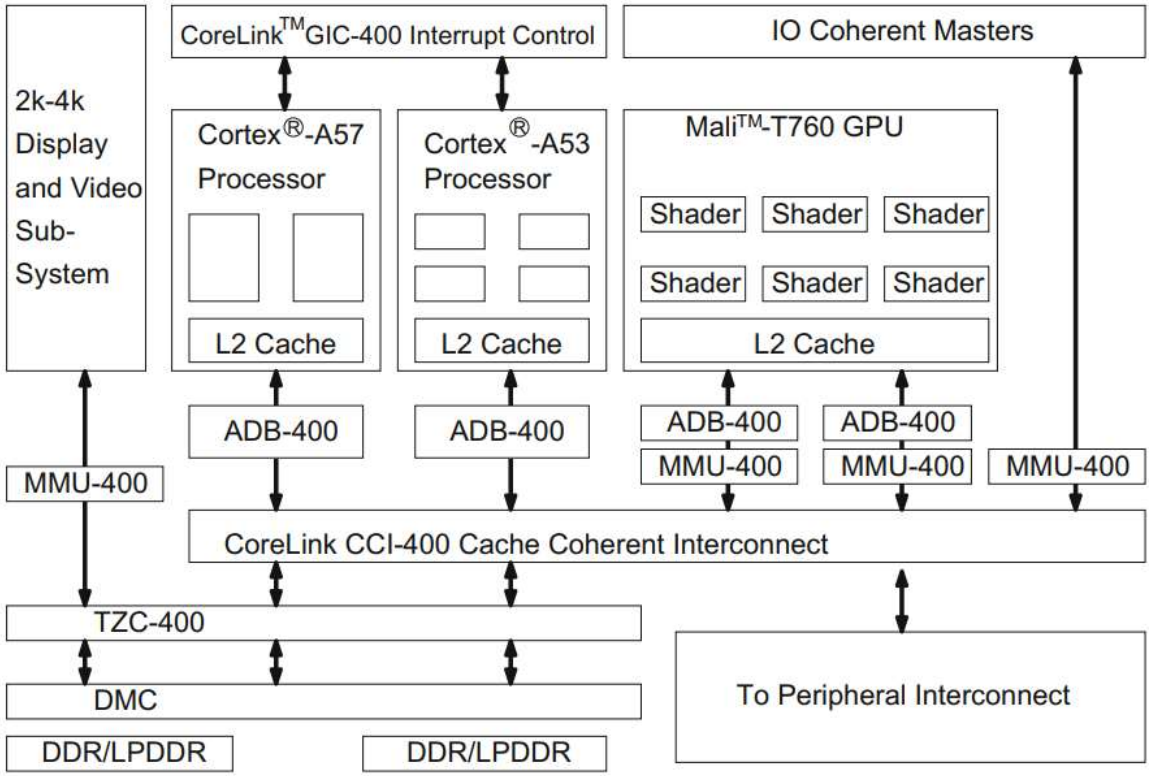


Fig. 13 ARM® big. LITTLE System on Chip (SoC)

4. Embedded System Hardware

4.3.3 *Reconfigurable Logic*

In many cases, full-custom hardware chips (ASICs) are too expensive and software based solutions are too slow or too energy consuming. Reconfigurable logic provides a solution if algorithms can be efficiently implemented in custom hardware. It can be almost as fast as special-purpose hardware, but in contrast to special-purpose hardware, the performed function can be changed by using configuration data. Due to these properties, reconfigurable logic finds applications in the following areas:

- **Fast prototyping**: Modern ASICs can be very complex, and the design effort can be large and take a long time. It is, therefore, frequently desirable to generate a prototype, which can be used for experimenting with a system which behaves "almost" like the final system. The prototype can be more costly and larger than the final system. Also, its power consumption can be larger than the final system, some timing constraints can be relaxed, and only the essential functions need to be available. Such a system can then be used for checking the fundamental behaviour of the future system.
- **Low-volume applications**: If the expected market volume is too small to justify the development of special-purpose ASICs, reconfigurable logic can be the right hardware technology for applications, for which software would be too slow or too inefficient.
- **Real-time systems**: the timing of reconfigurable logic-based designs is typically known very precisely. Therefore, they can be used to implement timing-predictable systems.

4. Embedded System Hardware

4.3.3 *Reconfigurable Logic*

Applications benefiting from a **very high level of parallel processing**: For example, parallel searches for certain patterns can be implemented as parallel hardware. Therefore, reconfigurable logic is employed in searches for genetic information, for patterns in Internet messages, in stock data, in seismic analysis, and many more.

Reconfigurable hardware frequently includes random access memory (RAM) to store configurations. We distinguish between **persistent** and **volatile** configuration memory**.** For ***persistent memory, information is retained even when power is shut off***. For ***volatile memory, the information is lost once power is shut down.*** If the configuration memory is volatile, its content must be loaded from some persistent storage technology such as read-only memories (ROMs) or flash memories at start-up.

Field programmable gate arrays (FPGAs) are the most common form of reconfigurable hardware. As the name indicates, such devices are programmable "in the field" (after fabrication). Furthermore, they consist of arrays of processing elements. As an example, Fig.14 shows the column-based structure of the Xilinx® UltraScale architecture. Some columns contain I/O interfaces, clock devices, and/or RAM.

4. Embedded System Hardware

4.3.3 *Reconfigurable Logic*

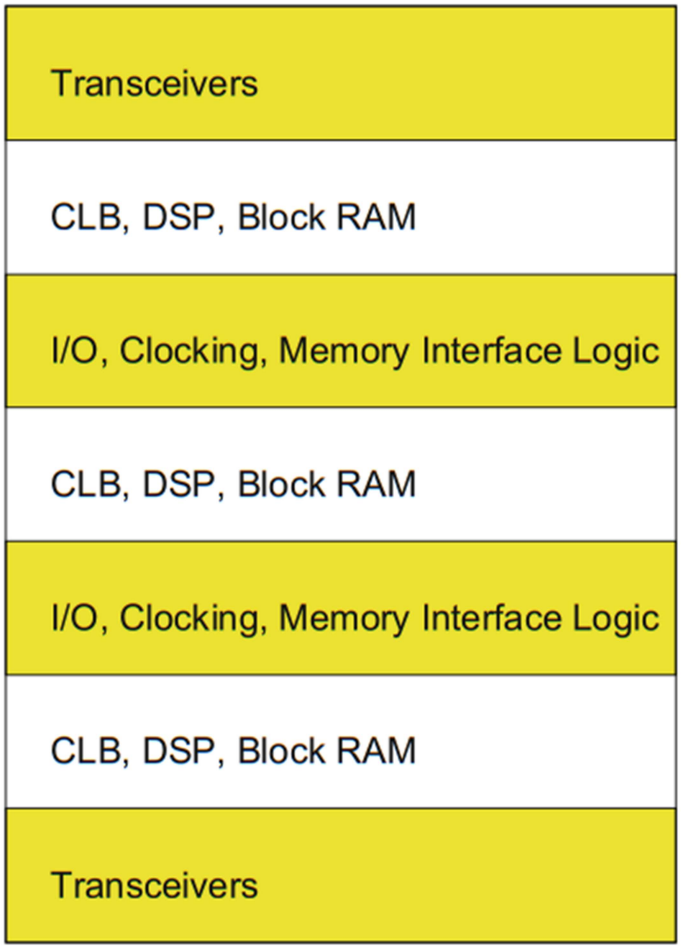


Fig 14 Floor-plan of column-based Xilinx® Ultra Scale FPGAs

4. Embedded System Hardware

4.3.3 *Reconfigurable Logic*

Other columns comprise **configurable logic blocks** (CLB)s, special hardware for digital signal processing, and some RAM. CLBs are the key components. They provide configurable functions. The architecture of Xilinx® UltraScale CLBs is shown in Fig. 15.

In this architecture, each CLB contains eight blocks. Each block comprises a RAM which is used to implement logic functions by a look-up table (LUT, shown in red), two registers, multiplexers, and some additional logic . ***Each LUT has six address inputs and two outputs***. It can implement any **single Boolean function of six variables** or **two functions of five variables** (provided that the two functions share input variables). This is the key means for achieving **configurability**.

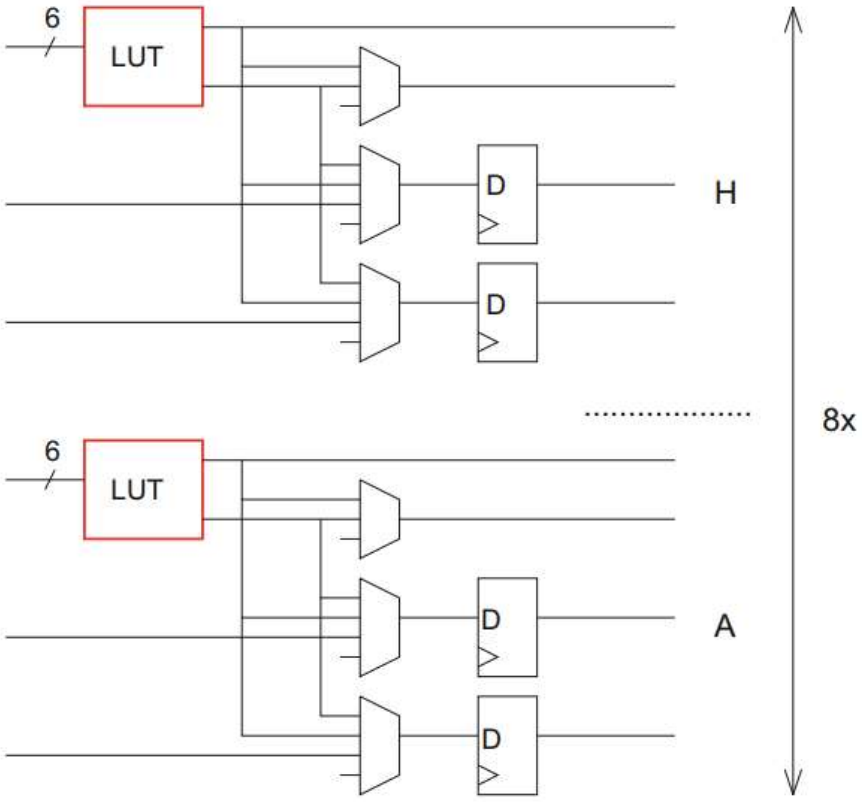


Fig. 15
Xilinx® UltraScale CLBs

4.3.3 *Reconfigurable Logic*

Recently the hard cores are available, for example, on Zynq Ultra Scale MPSoCs. They contain up to four ARM® Cortex-A53 cores, two ARM Cortex-R5 cores, and a Mali-400MP2 GPU processor. Typically, configuration data is generated from a high-level description of the functionality of the hardware, for example in VHDL. FPGA vendors provide the necessary design kits. Ideally, the same description could also be used for generating ASICs automatically. In practice, some interaction is required. Exploitation of the available parallelism typically requires manually parallelized applications, since automatic parallelization is frequently very limited. The parallelism offered by FPGAs is typically not fully exploited if all computations are mapped to processor cores. Overall, FPGAs allow implementing a huge variety of hardware devices without any need to create hardware other than FPGA boards.

Alternate providers of FPGAs include Altera® (see http://www.altera.com, acquired by Intel®), Lattice Semiconductor (see http://www.latticesemi.com), Microsemi (formerly Actel, see http://www.microsemi.com), and Quicklogic (see http://www.quicklogic.com).

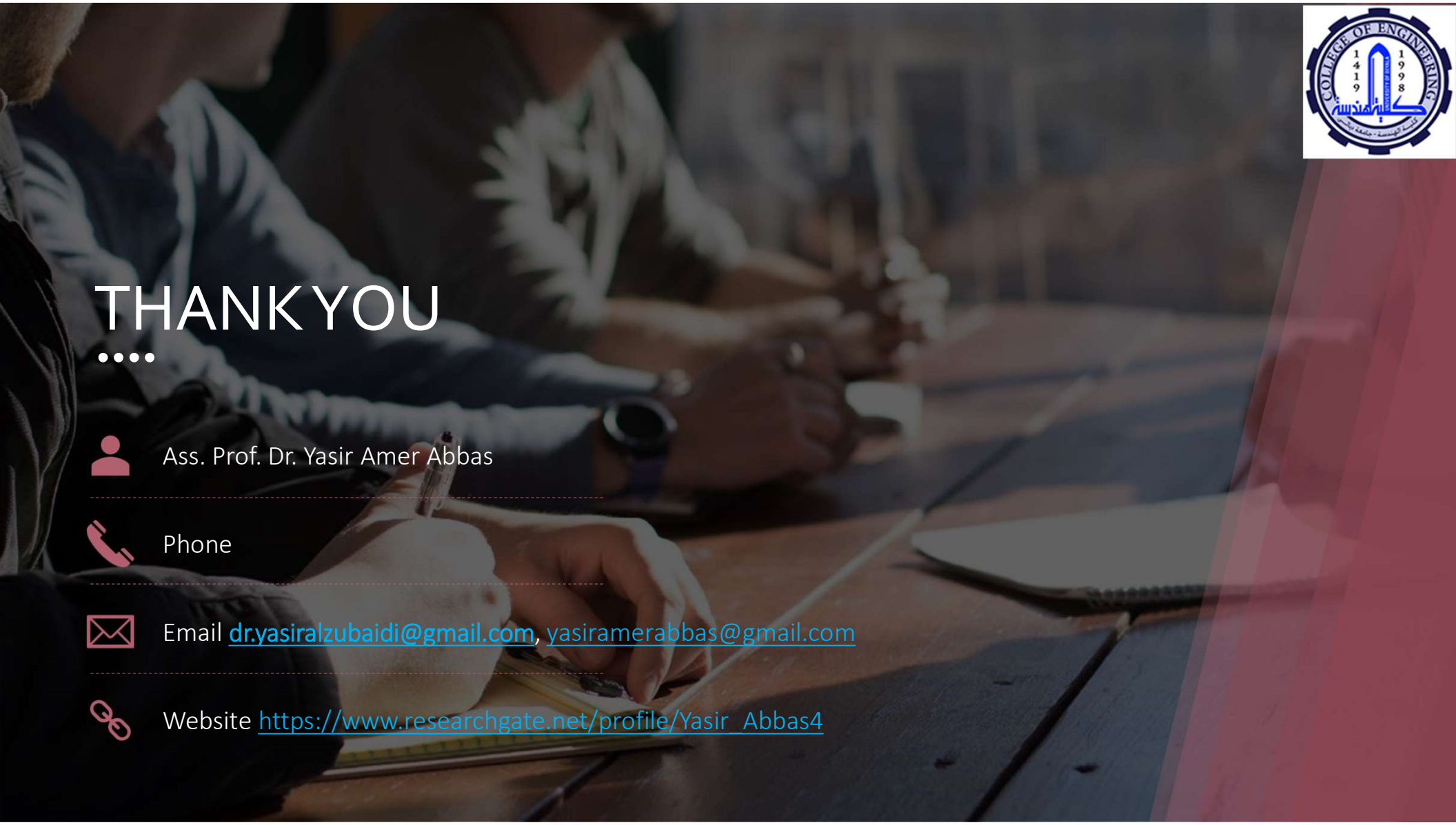
4. Embedded System Hardware

4.3.3 *Reconfigurable Logic*

- **Program memory:** It is a memory for storing the program required by DSP to process the data.
- **Data memory:** It is a working memory for storing temporary variables and data/signal to be processed.
- **Computational engine:** It performs the signal processing in accordance with the stored program memory computational engine incorporated many specialized arithmetic units and each of them operates simultaneously to increase the execution speed. It also includes multiple hardware shifters for shifting operands and saves execution time.
- **I/O unit:** It acts as an interface between the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals.

Examples: Audio video signal processing, telecommunication and multimedia applications.

SOP(Sum of Products) calculation, convolution, FFT(Fast Fourier Transform), DFT(Discrete Fourier Transform), etc are some of the operation performed by DSP.



THANK YOU

Ass. Prof. Dr. Yasir Amer Abbas

Phone

Email dr.yasiralzubaidi@gmail.com, yasiramerabbas@gmail.com

Website https://www.researchgate.net/profile/Yasir_Abbas4