

Chapter 2

Systems for Wireless Communication

The advent of second generation (2G), digital mobile communication networks for the mass markets had a significant impact on the use of mobile communication in the 1990s. Previously, the usage of mobile communication had been limited to business customers because of the high costs, whereas second (2G) and following (3G, LTE) wireless communication generations have been affordable for the masses. With the change of customers, the usage of mobile communication has broadened from pure mobile voice communication to infotainment and entertainment. This requires mobile handsets to support, in addition to the key components of voice and data communication, applications, like multimedia ones. The different structure and demands of these applications require different kinds of wireless communication protocols and standards which, in turn, has led to the incorporation of a hardware subsystem for each standard. This solution promises short-term success, however in the long term this principle is not expected to scale with a large number of supported communication standards. Finally, this has led to the vision of a Software Defined Radio (SDR) [6] which implements these standards in software to allow an easy upgrade and extension of the set of supported standards. It is commonly agreed that heterogeneous Multiprocessor System-on-Chip (MPSoCs) [7] are the best choice for the underlying platform to cope with the challenging demands of computational performance, energy efficiency, and flexibility, especially for wireless communication devices like SDRs.

This chapter first examines the applications executed on cellphones and smartphones separately for the three domains of wireless communication, multimedia, and general purpose. Based on them, the impact and constraints for the design methodology for wireless communication platforms are derived. The second part of the chapter discusses the underlying hardware platforms and components. Additionally, the specific influence of the platform and components on the design process is highlighted.

2.1 Applications for Mobile Devices

Applications for mobile devices differ significantly in their characteristics according to their domain. Therefore, they are discussed separately. Applications for wireless communications, with particular focus on physical-layer processing, are treated in greatest detail as the case study discussed in Chap. 8 addresses this domain.

2.1.1 Wireless Communication Domain

Within this area, targeted applications comprise all kinds of standards and protocols for voice and data communication. To achieve highest interoperability these are typically standardized by organizations like ITU [8], ETSI [9], and IEEE [10]. In addition, the application structure is defined according to the International Standard Organization Open Systems Interconnection Basic Reference Model (ISO/OSI Reference Model) [11] to simplify the design of wireless communication standards. However, modern standard implementations are not too strict about dividing the different layers, so that applied cross-layer optimizations soften the borders between adjacent layers.

A large variety of wireless communication standards have emerged, each addressing a particular range of user-level applications. The traditional classification of standards differentiates among Wireless Personal Area Networks, Wireless Local Area Networks (WLAN), Wireless Metropolitan Area Networks (WMAN), and Wireless Wide Area Networks. Figure 2.1 illustrates these four classes including examples and use-cases. Additionally, localization services like the Global Positioning System (GPS) are considered as a part of wireless communication systems.

The multimedia and wireless communication domains are converging initiated by technology advances, e.g., high performance mobile processor cores, as well as high-resolution displays and touchscreens for mobile devices. The result of this convergence is a class of *smartphones* that combine the functionalities of

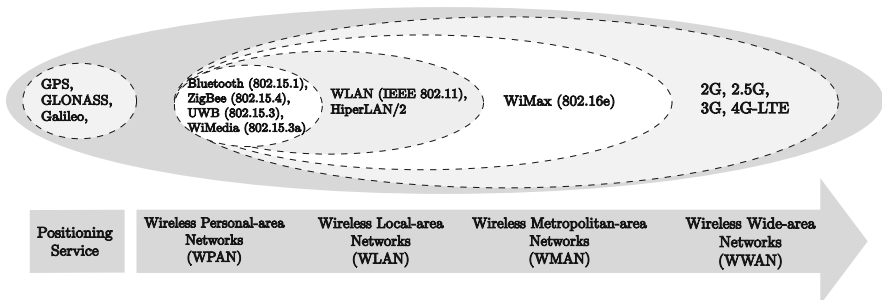


Fig. 2.1 Wireless communication networks

mobile phones and personal computers (PC) into a single mobile device. These devices support a wide range of different applications, each having individual connectivity demands. This requires the support of different wireless communication standards, e.g., Bluetooth [12] for wireless headsets, WLAN [13] for internet access, and 2G and 3G network connection for voice and data communication. Past and present designs cope with this challenge by incorporating one subsystem for each supported standard. For example, Apple's 3G iPhone [14] includes five subsystems for GSM/GPRS/EDGE (2G), WCDMA/HSDPA (3G), GPS, WLAN, and Bluetooth [15]. The addition of further subsystems to support additional wireless communication standards is not expected to scale in future. To cope with this issue, industry and research have opted for SDR [6], where different wireless communication standards are implemented in software allowing the reuse of hardware components. A case study [16] carried out by Infineon expects an SDR to outperform the traditional solution in terms of area and costs at five implemented standards. However, implementing even a single wireless communication standard is already a complex task, therefore the design of a complete SDR becomes quite challenging.

The development of wireless communication standards is dominated by the physical-layer processing, i.e., the lowest layer in the ISO/OSI reference model. This layer has a high computational demand (10–80 GOPS) and (mostly) hard real-time constraints have to be fulfilled. From the application perspective, the most severe constraints are *latency* and *throughput*. Failing to comply with these constraints will most likely lead to business failure.

The key elements of physical-layer processing are digital signal processing algorithms. These algorithms are typically characterized by a computationally intensive data-plane processing at high data rates predominantly controlled through parametrization. These data flow dominated applications are rather well structured in terms of *task graphs* or block processing, allowing the utilization of static schedulers and making task-level parallelism rather clear. The known task structure and data communication between different tasks can be easily captured in task graphs, e.g., Kahn Process Networks (KPN) [19] or Synchronous Data Flow (SDF) [20] task graphs. For specific task graphs, especially for the latter mentioned SDFs, a static schedule can be derived prior to run-time. Thus, deterministic behavior is ensured and no dynamic overhead occurs. Figure 2.2 exemplifies a task graph structure of a WLAN 802.11a receiver [17] implementation.

2.1.2 Multimedia Applications

The domain of multimedia covers a wide range of applications like audio, image, and video processing along with 2D and 3D graphic applications. Similar to the wireless communication domain, many standards coexist in the field, each having a particular optimization criterion like data compression or high quality.

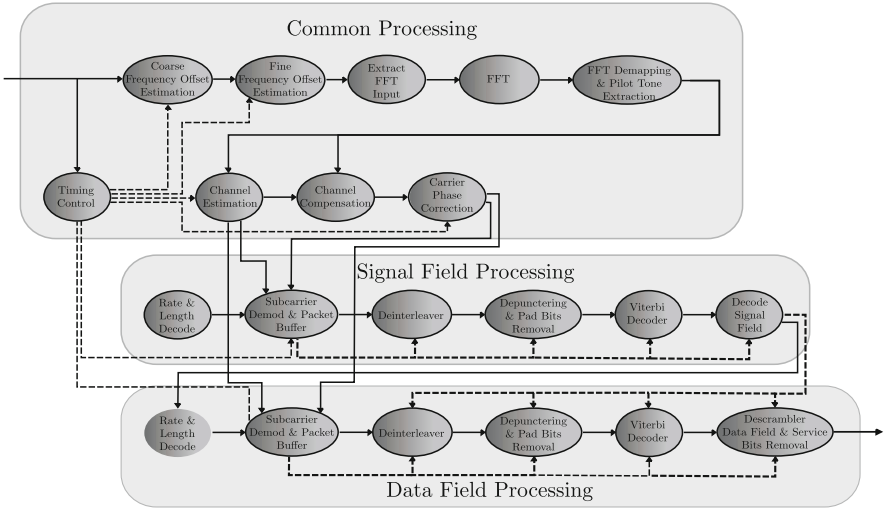


Fig. 2.2 Wireless communication task graph example: WLAN 802.11a receiver [17, 18]

Table 2.1 Computational and communication requirements of multimedia applications [21]

Typical configuration		On-chip communication requirements		Computational requirements in operations per second (GOPS)
Resolution	Frequency (Hz)	Pixels per second (Mpxl/s)	Bytes per second (MBps)	
720 × 480	60	20.7	82.8	31.1
1,280 × 768	60	59.0	236.0	123.9
1,920 × 1,080	30	62.2	248.8	186.6
1,920 × 1,080	60	124.4	497.6	373.2
1,920 × 1,200	60	138.2	552.8	414.7

Multimedia applications are characterized by a *high computational* demand along with *high communication* requirements between processor and memory. The key application in this domain is video processing with requirements including stringent real-time constraints as exemplified in Table 2.1. This especially applies to high-quality video applications, which include high throughput demands with challenging latency requirements.

These challenging demands of the latest multimedia standards along with the need for energy efficiency, in particular for battery-powered mobile devices, has led to highly specialized hardware accelerators and graphic-processing units [22]. These hardware components are optimized for performing typical 2D and 3D graphic processing operations, e.g., texture mapping and rendering especially vertex, geometry, and pixel shader calculations. The key characteristic of these algorithms is the *massive parallelism* of vector and matrix operations. Similar to ASIPs, these hardware architectures are especially tailored for the needs of multimedia applications. The design principle is to restrict flexibility to a minimum to achieve highest

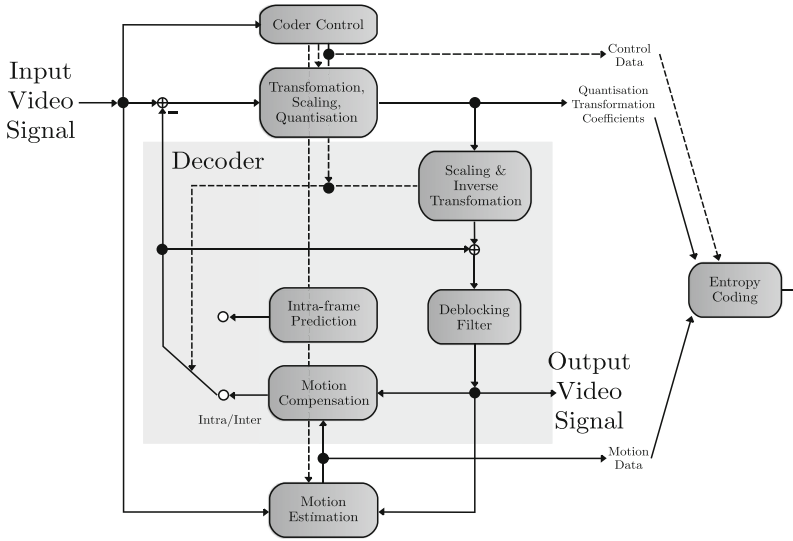


Fig. 2.3 Multimedia example H.264 task graph [23]

performance and energy efficiency. Despite the immense performance provided by such architectures, software development for them is extremely challenging. In addition, identifying the inherent parallelism within a particular algorithm is key and is mostly carried out by application experts and manual interaction.

Multimedia standards are mostly defined as task graphs (Fig. 2.3) like applications from the domain of wireless communication. The included control flow, e.g., the control overhead in H.264 decoding, leads to severe challenges in memory optimizations and data communication. Therefore, implementations commonly require special treatments to optimize the data communication.

2.1.3 General Purpose and Other Applications

Various kinds of applications are categorized under the term of general-purpose applications. Typical examples are text processing and web-browsing applications. Traditionally developed for personal computers (PCs), these are becoming increasingly popular even on mobile devices like smartphones. These applications are software-centric and make heavy use of operating systems (OSs), middleware layers, and other forms of hardware abstraction layers (HAL) such as hardware dependent software (HdS). Contrary to performance-critical parts, like physical-layer processing in the domain of wireless communication, they have a less dominant data plane processing and their computational complexity is rather low. On the other hand, control plane processing is much more severe because applications have to react on *nondeterministic* user interactions.

This leads to a complex control flow execution, which requires techniques for efficient execution, such as efficient implementations of jump and branch instructions as well as function and procedure handling. To accelerate them, well-known personal computer techniques, such as like branch prediction and superscalar architectures [24], are being increasingly adopted. This architecture trend is constantly narrowing the gap between general purpose processors within the embedded system and the personal computer market. Naturally, this opens new market opportunities for IP vendors from the embedded domain like ARM, MIPS, and Tensilica while moving into the direction of GPC. However, companies originating from the domain of personal computing, e.g., Intel, AMD, and VIA, have announced or are already are moving toward embedded systems [25].

In contrast to the previously discussed application domains, several description and development techniques for general purpose applications exist. However, the most common method is the classical textual design based on a high-level programming language based on C/C++ or Java. Other approaches like component-based software design [26] or the unified modeling language (UML) [27] provide graphical design entries for improved implementation efficiency.

2.1.4 Application Impact on Design Methodology

The rapidly increasing performance demands and limited available energy of battery powered devices, gives rise to an increasing energy-performance gap. Additionally, the need to jointly support various applications and their requirements is having a significant impact on the design methodology. *General purpose applications* demand flexible architectures to support a wide range of applications. Characterized by a dominant control path, software development relies on high-level programming languages along with operating systems (OSs), middlewares, and libraries. In contrast, applications from the domain of *wireless communication* and *multimedia* are implemented by highly specialized architectures and low-level software development. Applications of these domains are characterized by high computational demands in the data plane processing with relatively low control overhead.

In general, these various application requirements have significant impact on the design methodology of the two major components *software* and *hardware*. From the hardware perspective, the complexity and computational demand of modern communication standards requires rapidly increasing performance while preserving energy efficiency for future wireless communication devices. As the current technology scaling cannot cope with these requirements by itself, new approaches have to be considered [5]. An obvious solution is to apply parallelism, in terms of processing the application on multiple processing elements in parallel. In addition, the contradictory requirements of high computational power and energy efficiency require highly specialized hardware architectures. This has led to the common agreement that *heterogeneous MPSoC* platforms are the best candidate for such devices [28].

Unfortunately, the selection of heterogeneous MPSoC platforms has a significant impact and induces design challenges like:

- Partitioning of tasks to optimally exploit the inherent parallelism within a given application.
- This partitioning is tightly linked to the selection of the type and number of hardware components, which is a key question for assembling the hardware architecture.
- Performance evaluation can no longer be performed on the basis of a single isolated component. Instead, the interacting behavior of all system components requires a system-wide performance evaluation.
- New programming techniques and models need to be considered since, due to the heterogeneous nature of the platform a simple adaptation of known multiprocessor programming is not feasible.

In addition, the first and most important design objective is to achieve the performance requirements, mostly given in regard to *latency* and *throughput* constraints. These requirements, particularly when implementing the physical layer of a wireless communication standard, are characterized by stringent (hard) real-time constraints that have to be fulfilled. Otherwise, devices will most likely fail standard compliance tests, leading to business failure. Hence, the design methodology must incorporate techniques to efficiently evaluate whether the application-induced constraints are met or not. As late design changes tend to be more costly than early ones, such techniques should be applied as early as possible in the design process.

After discussing the application needs and their coarse-grained impact on the design methodology, the discussion now turns to detailed design aspects and the corresponding influence of each possible hardware component. Along with this, the impact on the design methodology is highlighted.

2.2 Hardware Platforms and Components

New design methodologies offering increased productivity in terms of design efficiency are indispensable for the development of future heterogeneous MPSoCs. For the comparison of different MPSoC platforms the following fundamental objectives and metrics can be defined.

Performance. Probably the most important design objective, the performance, is typically measured in terms of latency and throughput. Especially, meeting the performance constraints induced by applications is highly challenging but necessary for a successfully operating device.

Energy and Power Efficiency. Energy efficiency is one of the most severe design issues and platform differentiators. Especially, for mobile and battery powered devices energy efficiency is essential. Unfortunately, over the last years battery capacity has not been able to cope with the increasing performance demands, leading

to a growing performance-energy gap. This requires architectural innovations to increase the energy efficiency needed at present and definitely in the future. The metric *Millions of Instructions Per Second (MIPS) per Watt* typically defines energy efficiency [29]. Although this rather crude definition gives designers a first rough idea, it is unsuitable, as it is the required *energy per task* which matters. In the domain of wireless communication this metric can be expanded to the required *energy per decoded bit* or, within the domain of multimedia, to *energy per pixel*. The power efficiency classifies the power dissipation on the chip which influences the package and the layout of the final chip.

Cost. In general the total costs consist of the design costs and the initial manufacturing costs [30]. Whereas the design costs include the development of both software and hardware, the initial manufacturing costs comprise the mask and wafer costs as well as the initial packaging and testing. The dominating design costs are related to software and hardware development. These are reported for current design technologies (90 nm) to be in the region of 10–100 million USD with an expected increase of 50–100% per shrink in the process generation. Whereas in the past hardware-development costs claimed the major portion, the increasing use of programmable components has led to rapidly increasing software costs [30]. Latest market studies of MPSoC design report them to be at the same level. In addition, chip mask production has become increasingly expensive and is typically in the range of multiple million USD for each mask iteration.

Flexibility. In contrast to the previously discussed objectives and metrics, flexibility cannot be simply given as a single value. Flexibility defines the capability to execute a specific functionality on a particular processing element. This metric is of vital importance especially when designing SDRs [6]. Additionally, flexibility has the advantages of enabling short time-to-market and extending the lifetime by applying software updates and bugfixes. It is closely related to portability, which defines the ease of porting a certain functionality from one platform to another. Portability can be defined as the inverse of the porting effort [31] which, in turn, directly relates to flexibility.

These objectives help to guide system architects in their design decisions to find the optimal design. However, the complexity and short time-to-market along with the discussed requirements put a particular pressure on the development of such MPSoC platforms. Therefore, new design methodologies have to be considered to minimize the required development effort and costs. Here two fundamental design concepts, namely component-based design (CbD) [32] and platform-based design (PbD) [33], have been envisioned and found major acceptance.

MPSoC design: Evolution rather than Revolution. According to the component-based approach, the complete platform is assembled from in-house or external IP components, e.g., processor cores, communication architectures, memories, and many other IP components. The key to the efficient use of this design principle is a unified interface definition to connect arbitrary IP components. These interfaces are mostly bus or Network-on-Chip (NoC) centric, like the interfaces of the

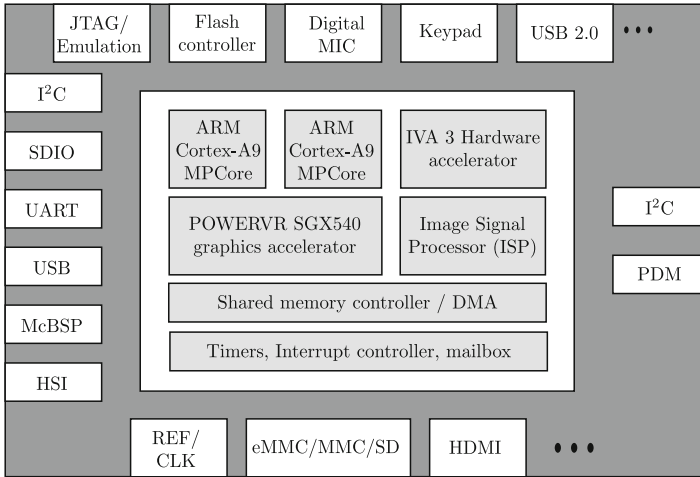


Fig. 2.4 IP block structure of the TI OMAP44x platform [38]

AMBA bus [34] or the IBM CoreConnect [35]. These have been standardized or evolved to a de facto standard by wide utilization. Based on this design methodology a large variety of companies have established a successful IP business, among them processor IP vendors like ARM, MIPS, and Tensilica as well as communication architecture IP providers like Arteris [36] as well as above-mentioned IP vendors like ARM and IBM. An example IP-component structure for TI's OMAP [37] platform is sketched in Fig. 2.4.

This Cbd inherently ensures the high reuse of components over different platforms as they are separated by well-defined interfaces. Because of growing complexity, the average number of IP components an MPSoC platform consists of has increased from 25 in 2006 to 28 in 2007 and 33 in 2008 [39]. Further predictions expect an increase over the next years, already reaching 72 IP components in an average platform design by the year 2012.

With the aid of such IP components, PbD has proved to be highly suitable to quickly obtain modified platforms from a base one. The major element is the restriction of the design space by reducing flexibility, which simplifies and shortens the development cycle significantly. This design methodology has been successfully applied to especially address a specific market segment, e.g., the areas of wireless communication and multimedia. Prominent examples are TI's OMAP platforms for wireless communication devices and Philips Nexperia [40] platforms for multimedia applications.

The development of each platform is based on a construction kit. For each market segment a particular set of IP components is selected and connected. For example, the OMAP331 targeting the low-cost segment consists of an ARM926EJ-S processor core with a few surrounding peripheral devices. The high-cost segment is addressed by TI's OMAP3430 [37] platform that includes a more powerful ARM

Cortex-A8 processor, an IVA 2+ graphics accelerator, a POWERVR SGX graphics core [41], a dedicated image signal processor (ISP), and various other peripheral devices.

Apart from the business success of such platforms, this design methodology bears some hidden traps and risks [42, cf. 7.2]. The key risk is that system architects enter the design cycle biased and do not question design decisions related to the preexisting software or hardware IPs. In the end this can lead to false design decisions that decrease performance or increase energy consumption. In contrast, starting designs from scratch without reusing pieces of existing platforms is also no option when considering the tight time-to-market constraints. Therefore, a suitable design methodology requires a mixture of both extremes and demands strong design discipline. Hence, Bailey et al. [42] propose that all design options should be considered when developing a modified platform virtually starting with a blank sheet of paper, but characteristics, prior experiences and reuse of existing IP components can be incorporated to enhance the design process and the final platform.

As a large variety of different components exists, the rest of this section discusses each particular group of components separately and highlights the impact on the design methodology. However, it should be noted that the most essential issue in MPSoC design is the interwoven behavior of all the components and not that of a single isolated component. For example, a high performance processor core cannot fully exploit its capabilities if either the communication architecture or the memory subsystem is too slow to deliver the necessary data to be processed. Such issues cannot be evaluated in an isolated fashion because they only occur when investigating the system-wide performance.

2.2.1 Processing Elements

The class of processing elements ranges from highly flexible general purpose processors (GPPs) to dedicated hardwired accelerators, optimized for a particular function. Lately, the demand for postfabrication flexibility has led system architects to increasingly use flexible and programmable components like general purpose processors, digital signal processors (DSPs), and application-specific instruction-set processors. Consequently, the amount and the importance of software are steadily increasing. Already today software has become one of the most critical pieces in system design [43], consuming a significant amount of the overall budget. With the increasing introduction of heterogeneous MPSoCs, various software design methodologies need to be considered jointly ranging from high-to low-level software constructs.

The class of the processing elements can roughly be classified into the following groups.

- General Purpose Processor (GPP)
- Digital Signal Processor (DSP)
- Application Specific Instruction Set Processor (ASIP)

- Reconfigurable Application Specific Instruction Set Processor (rASIP)
- Field Programmable Gate Array (FPGA)
- Application Specific Integrated Circuit (ASIC)

General Purpose Processors offer high flexibility and are hence utilized for arbitrary applications like control and user-level applications. Commonly, application development is conveniently carried out in high-level programming languages, e.g., C/C++ and Java. Often an operating system (OS) is supported and software development is abstracted by HAL or other middlewares from low-level hardware features. This shields software design from the underlying hardware by means of abstraction, permitting to concentrate on the pure application development.

Digital Signal Processors are especially tailored for the common characteristics and operations of digital signal processing algorithms. These processors exhibit special instructions to efficiently perform operations common to these algorithms, e.g., multiply accumulate, add-compare-select, and Galois field instructions [44]. The latest DSP architectures provide increased parallelism by means of Very Long Instruction Words [45], Single-Instruction Multiple-Data [46], and superscalar [47] hardware features. Because of the high-performance and low energy-consumption demands in the domain of wireless communication, fixed-point DSPs are still the first choice even after the introduction of floating-point DSPs [44].

Application Specific Instruction Set Processors are specially developed for a specific application. In general, the design of an ASIP follows the guideline of minimizing flexibility to maximize energy efficiency, area efficiency, and/or performance. Today, the class of ASIPs covers a wide range of different approaches and architectures. Tensilica's approach [48] enters the design process with the Xtensa processor core as a base architecture and allows further customization of this template with respect to the addressed application. Other approaches support ASIP development based on an Architecture Description Language (ADL), e.g., LISA 2.0 [49] or Expression [50]. These ADL-based approaches do not restrict designers in their decisions to support full architectural design space exploration. Contrary to GPPs, application-specific features cannot be easily addressed by compilers. Therefore, ASIPs typically require low-level software development to exploit the specific features. However, there are promising approaches to generate the software tool-chain including compiler, assembler, and linker for the ASIP [51, 52] with reasonable performance.

Reconfigurable Application Specific Instruction Set Processors extend the concept of ASIPs further by combining the base processor with a reconfigurable fabric based on FPGAs [53]. This combination of a fixed and a reconfigurable hardware architecture promises high performance with increased flexibility to adapt the designed processor to different applications. Compared with the previously discussed ASIPs, the reconfigurable part adds postfabrication flexibility. Already a few architectures [54] and design methodologies [55, 56] exist, highlighting the potential of such architectures. However, this research field is relatively new and is expected

to have high potential in the future. Besides the earlier-mentioned issues for ASIPs, additional hardware description language (HDL) programming needs to be included to program the embedded FPGA.

Field Programmable Gate Arrays are reconfigurable processing elements. Based on the capability to reconfigure the functionality after manufacturing, these components provide a particular postfabrication flexibility. The utilization of such devices has a strong impact on the design process, because FPGA devices are traditionally programmed in hardware description languages, e.g., VHDL [57] and Verilog [58]. Therefore, adding an FPGA to a platform changes the design process to a mixed software and hardware development. However, its flexibility compared to ASICs is achieved at the expense of decreased performance and increased energy consumption, but offers the possibility of reprogramming and bugfixing in the field.

Application Specific Integrated Circuits are specially tailored for a given algorithm or application. With the functionality fixed, only minor configuration can be applied after fabrication. Mostly this configuration is limited to the setting of algorithmic parameters, e.g., the filter coefficients of an FIR filter. In contrast to the restricted flexibility, energy efficiency and performance are relatively high. This leads to an integration of such processing elements in the performance-critical parts of a design. The traditional design focuses on known hardware design methodologies like Register Transfer Level (RTL), modeling with logic synthesis on standard-cell libraries, or full-custom design on transistor level.

Summarizing the common use of these processing elements, wireless communication and multimedia algorithms, as proved in the past, can be efficiently implemented on specialized hardware. Dedicated hardwired accelerators (ASICs) are especially tailored for a particular algorithm, whereas DSPs are optimized to the common characteristics of such algorithms, e.g., multiplications, multiply accumulate, and add-compare-select. Application Specific Instruction Set Processors (ASIPs), like those proposed by Wehn et al. [59] or SODA [2], are specialized processor cores which have been specially developed for a particular algorithm or multiple ones. The key principle of ASIPs is to minimize the provided flexibility to increase performance and to minimize overheads in terms of area, power and energy consumption. To incorporate such specialized architectures, software development cannot follow the general-purpose approach, as current high-level language compilers can hardly exploit such features optimally due to their highly irregular structure [31,60]. However, research focuses on this issue and promising approaches exist in literature [51,61–64].

In contrast to specialized processing elements, general purpose applications require a higher degree of flexibility. Hence, GPPs are typically utilized for their execution and the latest techniques and architectures from the personal-computer domain are increasingly being applied to mobile devices. For example, ARM Inc. has just recently announced the ARM Cortex-A8 processor core as their first super-scalar processor core. Additionally, multicore processors like the ARM Cortex-A9 can already incorporate up to four cores within a single entity.

So far only processing elements have been considered. However, with the increasing parallelism in future platforms, data exchange between the processing elements is becoming another key issue. In general, to transfer data from one element to another, a communication architecture and storage elements are necessary. Recently with the increasing number of interacting components, the principle of bus-based communication architectures has gradually tended to become the bottleneck of the complete system. Therefore, the latest research in this domain has proposed more complex communication networks subsumed under the term Network-on-Chip (NoC) [65].

2.2.2 Communication Architectures and Memory Subsystems

Despite the vast research and many publications within this domain, a precise definition of NoCs is typically not given [66]. The OCP-IP consortium defines the term NoC in a rather generic fashion as *a communication network that is used on chip* [67]. This generic definition allows further differentiation of NoC architectures under the key aspects of:

- *Switching policy*: circuit-switching and packet-switching.
- *Topology*: point-to-point, bus, hierarchical bus, crossbar, 2D-mesh, 2D-torus, 3D-torus, customized for the addressed application, etc.
- *Routing*: deterministic fixed routing and dynamic adaptive routing.
- *Quality-of-Service*: best effort and guaranteed throughput.
- Testing and fault-tolerance.

When dealing with embedded systems, research about NoCs has to adhere to the special demands of this domain in terms of cost, power and energy efficiency [68]. Similar to the application specific processing elements like DSPs, ASIPs, and rASIPs, customized application-specific NoCs achieve superior performance in terms of latency, throughput, area, power and energy efficiency by restricting the flexibility [67]. However, the highly irregular topologies of these communication architectures increase the effort required for wiring and layout. As this book focuses on early design space exploration of heterogeneous MPSoC platforms, interested readers are here referred to [69] for a detailed discussion of available Network-on-Chip architectures and design methodologies.

The general design approach of CbD, treats a memory subsystem as a single hardware IP component due to the highly regular structure that is attached to a particular communication architecture and used to exchange data. The memory portion in modern MPSoC platforms is tremendous and considered to be in the range of ~60% of the complete area [70]. Therefore, area and energy consumption can be significantly reduced by designing efficient memory architectures. A classical hierarchical memory system as illustrated in Fig. 2.5 attaches the processor core directly to a fast scratchpad memory or cache which is further connected to a larger memory and finally over I/O devices to external memories like hard disks and flashcards.

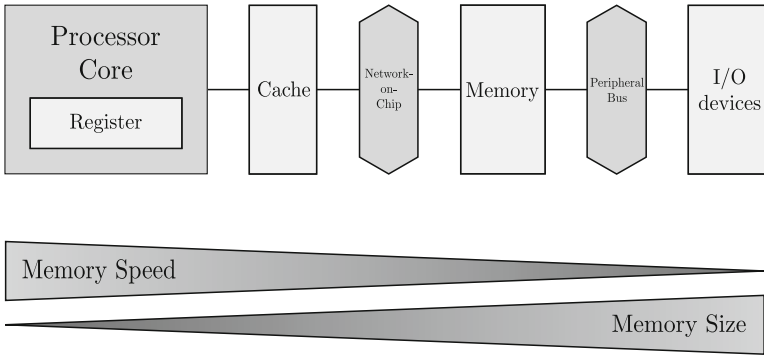


Fig. 2.5 Memory architectures of common processor cores [24]

With respect to early design space exploration, the impact of the memory subsystem has to be evaluated in terms of size and performance. Whereas, the memory size has a significant impact on the required chip area, the performance characteristic depends on the communication between processor and memory. This includes the underlying communication architecture and has to consider the occurring memory access patterns. Therefore, design methodologies have to support designers in performance evaluation of the memory subsystem together with the communication architecture. Key objectives are to minimize the required memory size to reduce area and energy consumption, as well as to increase performance characteristics in terms of latency and throughput.

2.2.3 Hardware Architecture Impact on Design Methodology

The chosen hardware architecture heavily affects whether the application performance constraints are met or not (Sect. 2.1) and how the platform behaves with respect to the formulated objectives (Sect. 2.2). An accurate evaluation of the performance and the objectives can only be done when based on the complete platform and not on the investigation of a single component. However, selecting each processing element can have significant impact on the design methodology.

Looking at the programmable part of the architecture, GPPs execute general purpose based applications that are typically developed in a high-level programming language including the use of operating systems, middlewares, and hardware abstraction levels. Conversely, performance critical applications execute on more application-specific architectures like DSPs, ASIPs, or even weakly programmable devices [71]. In general, such specialized architectures require hand-optimized low-level software implementations. Unfortunately, this influences the design methodology, so that time and cost intensive software development [72] is certainly required to achieve the necessary performance. Therefore, prior to the implementation step,

system architects *must* evaluate that the implementation of software and hardware satisfies the addressed performance requirements. Otherwise late design changes, e.g., exchanging the type of a processor core, will result in high time and cost investment as optimizations need to be applied twice [73]. This demands a sophisticated evaluation methodology for the early investigation of different design decisions leading finally to the implementation of both software and hardware. Hence, a joint hardware/software codesign including different software development techniques is indispensable.

In addition to the use of software-centric processing elements, performance-critical parts might require less flexible processing elements due to high computational requirements and/or low flexibility demand. In such cases components like ASICs, FPGAs, but also rASIPs should be considered. These processing elements extend the software-centric design flow to also incorporate traditional hardware-design methodologies. This joint consideration of software and hardware makes a mixed hardware/software codesign essential and tends to be complex.

Besides the impact of the processing elements, the impact of the communication architecture must not be neglected. In the high-performance computing domain more general and regular NoCs are selected, whereas in the area of embedded systems customized NoCs dominate the market. These NoCs are optimized for the particular needs of one or multiple applications and restrict the flexibility by limiting the number of physical links, which in turn restricts the mapping space of the application. Therefore, the development of these application-specific communication architectures requires a deep knowledge of all the other hardware components and the addressed applications. This leads to the necessity for a joint design method, including the investigation of arbitrary communication architectures.

In summary, the challenging performance requirements and objective constraints force system architects to utilize heterogeneous MPSoC platforms, which, in turn, significantly affects the software design methodology for such platforms.

2.3 Summary

So far this chapter has highlighted the impact of applications, hardware and software decisions on the design methodology. Driven by the complexity and requirements of future applications system architects are increasingly using heterogeneous MPSoC platforms. These platforms are commonly assembled from different IP components that can be roughly grouped into the classes of processing elements, communication architectures, and memories. Based on the selected processing element used to execute a particular functionality, hardware/software codesign becomes necessary. This becomes a key challenge, especially when utilizing application-specific and optimized processing elements. In addition, different software-development schemes have to be used jointly to optimally exploit the features of the underlying hardware.

The range of applied software development has to consider all possibilities, starting from low-level Assembly implementation up to high-level programming languages and the use of operating systems, middlewares, and other HAL.

The various challenges in designing such MPSoC platforms demand a structured design methodology. The next chapter discusses the principles of what is traditionally considered *design space exploration*.



<http://www.springer.com/978-1-4419-8152-3>

Multiprocessor Systems on Chip
Design Space Exploration
Kempf, T.; Ascheid, G.; Leupers, R.
2011, XIX, 189 p., Hardcover
ISBN: 978-1-4419-8152-3