Partitioning

- The term "partitioning" in the context of ESL design refers to the process of subdividing an initial specification for a system, which is generally defined by a monolithic natural language document, executable specification, or legacy design, or a combination of all three, into a set of potentially concurrent cooperating processes.
- Each of processes may be described by documents, executable models, or legacy designs, or a combination of these forms, and of assigning them to a set of more or less abstract resources, representing processors for software, silicon area or IP blocks for hardware, communication channels, and storage resources.

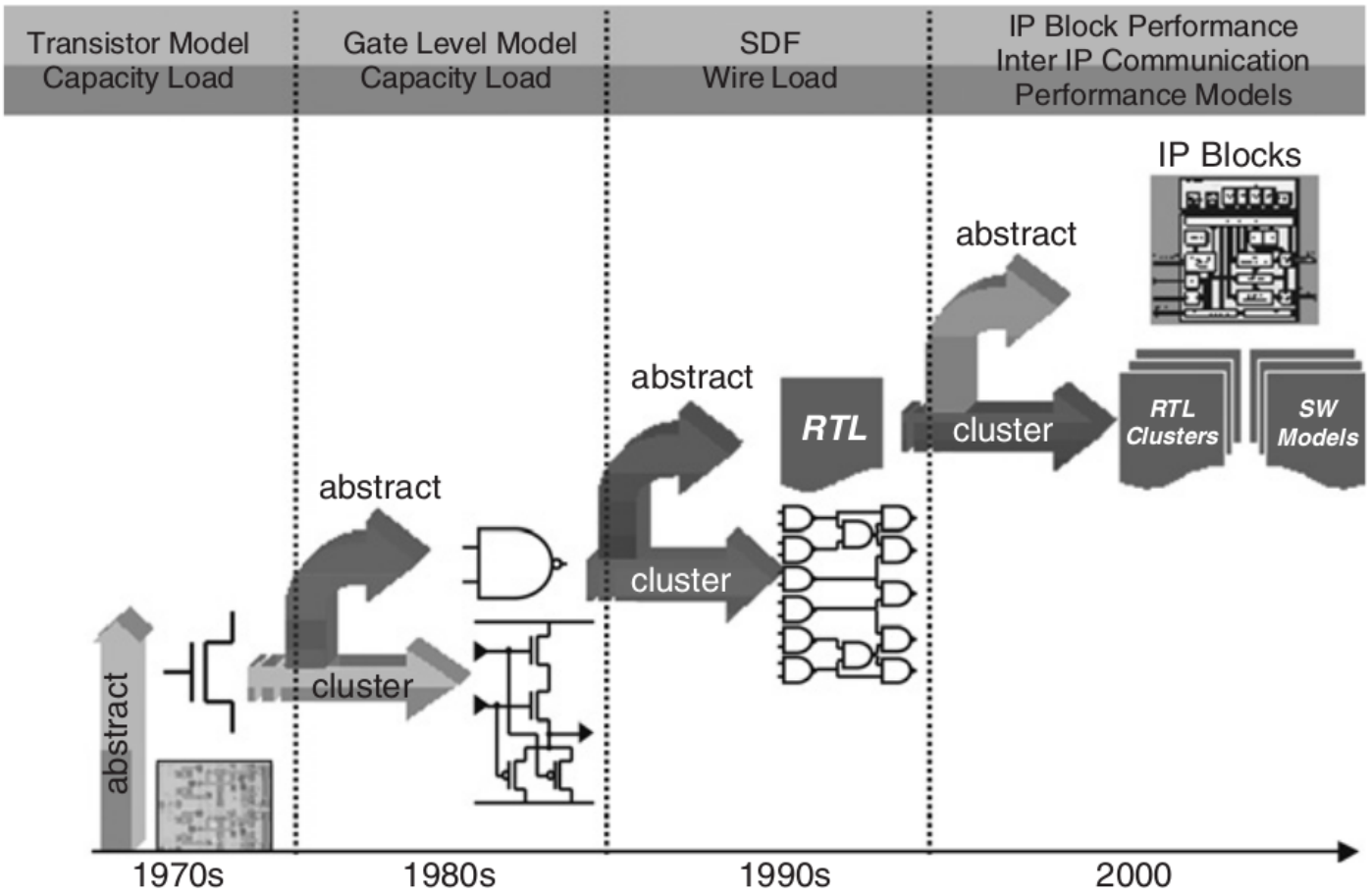
Partitioning

- Partitioning also speed up the overall design process by allowing teams to work in relative independence
- By introducing interfaces, the team can dramatically simplify the verification of each partition in isolation.
- Partitioning also reduces optimization opportunities by introducing barriers that can be either very expensive or impossible to cross.

Partitioning

- Divide and conquer paradigm.
- A sequence of two steps has traditionally been used to cope with the increased complexity and enhance designers' productivity: abstraction and clustering.
- Abstraction means describing an object using a model where some of the low-level details are ignored.
- Clustering means connecting a set of objects at the same level of abstraction to get a new, more complex object.

Partitioning - Clustering



Partitioning - Platform

- A platform is a single abstract model that hides the details of a set of different possible implementations as clusters of lower-level components.
- The example of platform is a family of microprocessors, peripherals, and bus protocols (Zynq platform) .
- The platform allows developers of designs at the higher level, called applications, to operate without detailed knowledge of the implementation (the pipelining of the processor or the internal implementation of the UART).
- At the same time, it allows platform implementers to share design and fabrication costs among a broad range of potential users, broader than if each design was a one-of-a-kind type.

Partitioning - Platform

- The relationship between an application and elements of a platform is called a mapping.
- This exists, for example, between logic gates and geometric patterns of a layout, as well as between RTL statements and gates.
- At the system level, the mapping is between functional objects with their communication links, and platform elements with their communication paths.
- Mapping at the system level means associating a functional behavior (e.g., an FFT or a filter) with an architectural element that can implement that behavior (CPU or DSP or piece of dedicated hardware).

Partitioning - Steps

- The first, and probably the most difficult, step is the identification of functional components from a "monolithic" specification, that is, one in which the final decomposition into implementation domains is not defined.
- The second step is the definition of the target architecture, while taking into account cost constraints and expected power, communication, storage, and performance requirements imposed by the specification.
- The third step is the assignment, again guided by the results of pre-partitioning analysis, of functional units to architectural units ("mapping"), while taking into account capacity, power, and performance constraints.

Partitioning - Steps

- The fourth step is the detailed definition of the interfaces between the various partitions, while within each partition communication is handled by the synthesis or manual implementation step that follows.
- In principle, any of the aforementioned steps could be automated. However, they become progressively more difficult as we move upward from the bottom.

Partitioning - Functional Decomposition

- To perform an efficient and effective partitioning of the system, one needs to start from a specification model that is as devoid as possible of implementation details, to enable the best optimization opportunities in selecting the architecture configuration and the mapping.
- For this reason, the preferred input to partitioning is a purely functional representation, annotated with constraints on cost, performance, power, and so on that restrict the mapping freedom.
- Furthermore, this specification must exhibit enough application-level parallelism to exploit the concurrency offered by the architectural platform (e.g., among processors, IP, and dedicated hardware).

Partitioning - Functional Decomposition

- These requirements are currently met by one of two mechanisms.
- One that is well entrenched in current design practice and well-supported by available tools is to use a functional concurrent executable specification language.
- The other is to start from a sequential language and automatically extract parallelism.
- The second approach is much more difficult, but at the same time much more promising because it can use monolithic executable specifications and legacy code that are aimed at a less concurrent implementation platform.

Partitioning - Architecture Description

- An architecture is usually viewed as a collection of components that provide a means to implement some kind of behavior.
- In its most basic form, each component in an architecture description can be identified with a real device, and the netlist represents the physical interconnections between those devices, such as a bus or a point-to-point communication.
- Elements in the architecture can be classified based on what kind of behavior they can implement and are usually divided into three broad classes: computation, communication, and storage.
- Such an architecture description is very close to a hardware implementation of the system, and can sometimes be viewed as an abstract schematic of the hardware design.

Partitioning - Architecture Description

- In other, more complex cases, the architecture description also includes elements that cannot be immediately associated with a physical device.
- Examples of these elements are a network protocol to be used over a communication device, the particular instruction set that is interpreted by the CPU, the arbitration policies used to manage the access to shared resources, and so on.
- The software architecture, representing the organization of tasks and layers that implements part of the system behavior, and possibly including an RTOS, should be considered when a detailed architecture description of a product is needed.
- These complex specifications, especially when the description can be tailored to specific cases and reused in several designs, are also referred to as platforms.

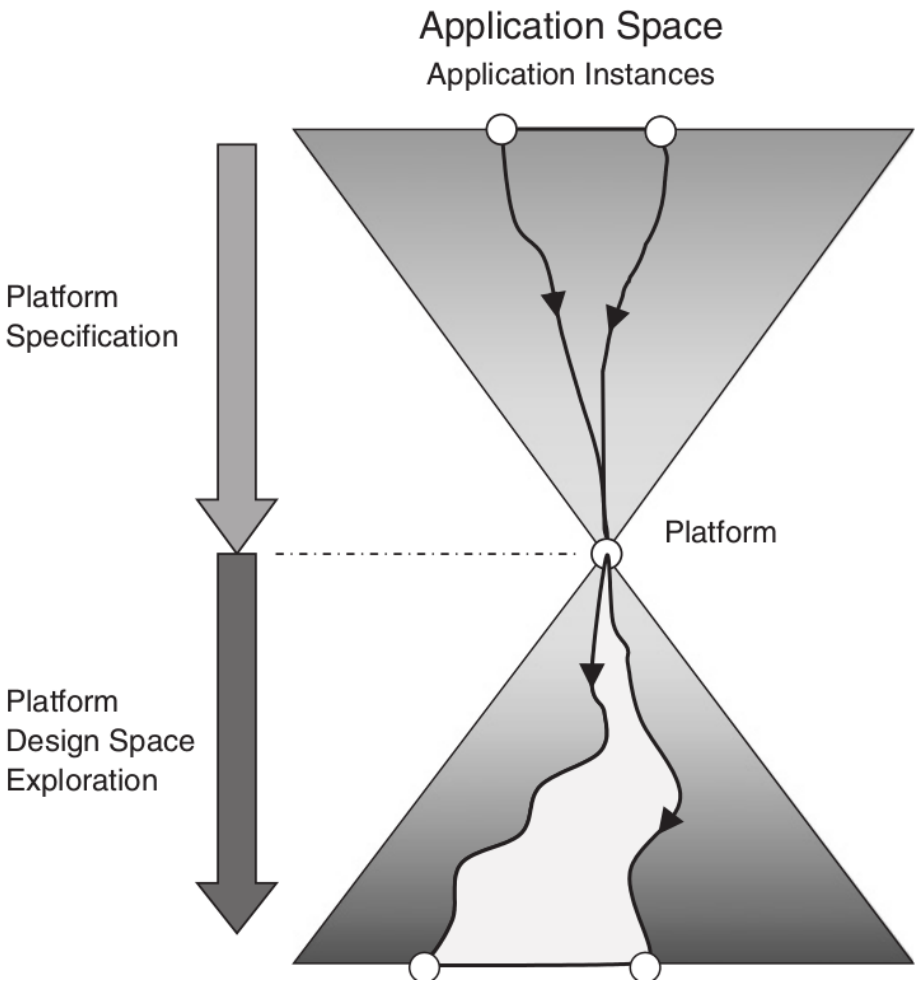
Partitioning - Platforms

- In most practical cases, the architecture for a new system is derived from exist- ing architectures of previous products.
- In fact, the need to meet time-to-market constraints forces designers to base the development on past experience and pre- designed components.
- It is not uncommon that several different designs share the same base architecture, either because they are different implementations of similar specifications (derivative designs for different markets) or because they have similar requirements.

Partitioning - Platforms

- In general, given an application domain, the number of good solutions to its main requirements is limited.
- Therefore, it is worthwhile to spend more time and money in the development of an architecture that best captures these good solutions because its design cost can be spread over a greater number of end applications. We call this architecture, or set of architectures, a platform.

Partitioning - Platforms



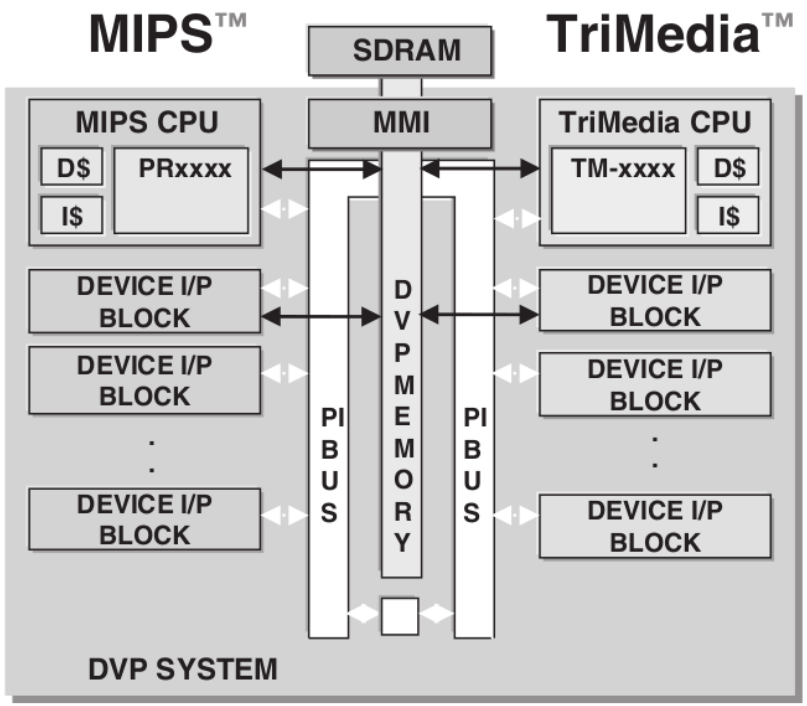
The NXP Semiconductor Nexperia hardware-software platform

GENERAL PURPOSE RISC PROCESSOR

- 50 to 300+ MHz
- 32-bit or 64-bit

Library of Device Blocks

- Image coprocessors
- DSPs
- UART
- 1394
- USB



VLIW MEDIA PROCESSOR:

- 100 to 300+ MHz
- 32-bit or 64-bit

Nexperia System Buses

- 32-128 bit

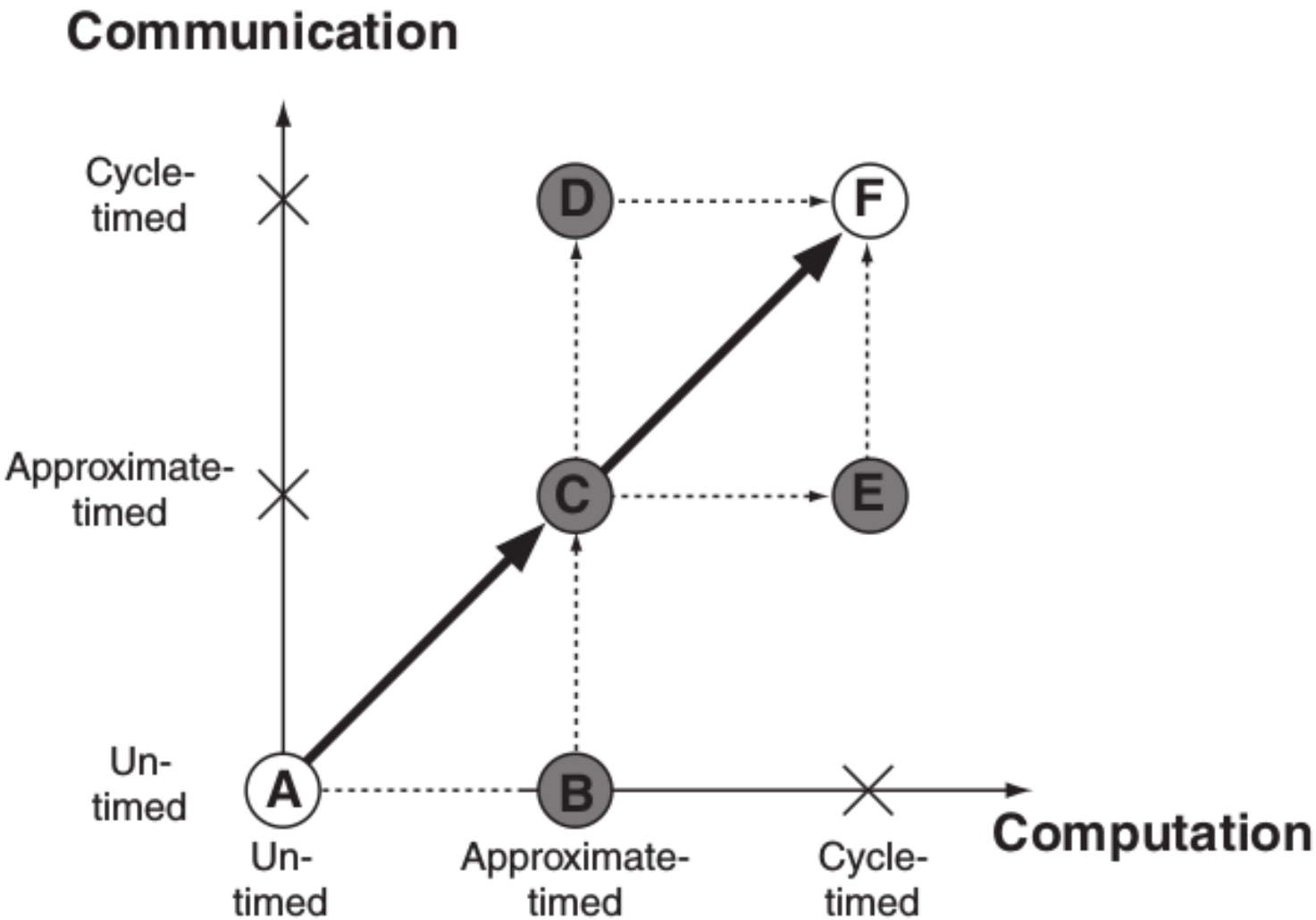
Partitioning - Assignment

- Partitioning a concurrent application onto architectural modules can be performed in two main ways:
  - By successive refinement steps that use a single multilevel modeling mechanism and add sufficient details at each step to transform the functional model into the mapped one.
  - By using an explicit mapping notation that is used by tools both to derive refined analysis models and guide synthesis steps.

Partitioning - Refinement-Based Methods

- One approach to defining the partitioning procedure is to introduce additional information in the form of refinements of the model abstraction by adding or making more precise the specification information.
- Two independent aspects, communication and computation, are most important in this approach.

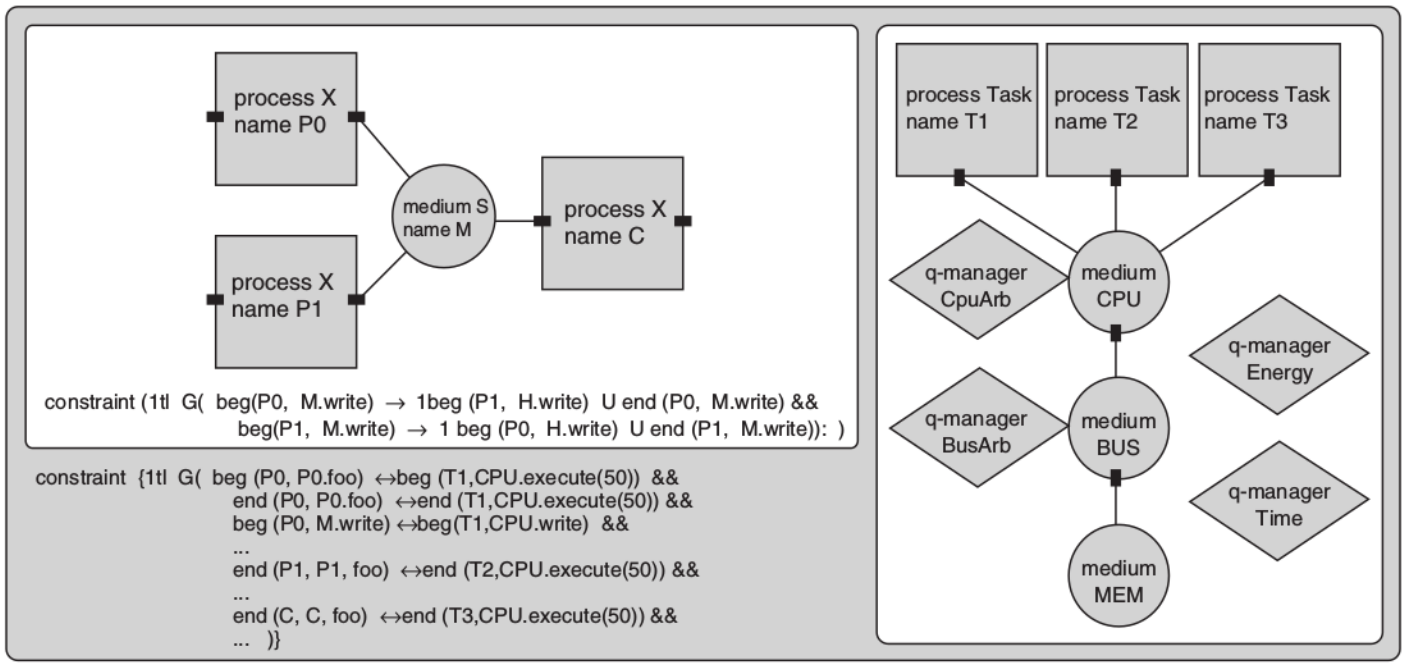
Partitioning - Generic Abstract Models



Partitioning - Refinement-Based Methods

- The most abstract level describes an untimed functional model A, also commonly understood as a specification model.
- Adding architectural information and the correlated functional timing behavior produces the refined model B, called a component assembly model.
- The bus arbitration choice allows one to introduce a first notion of real communication timing and results in the bus arbitration model C.
- From this model, the designer could choose two ways to reach the final goal.
- Either way, through the behavioral bus-functional model D or the cycle-accurate computation model E, leads to the final RTL implementation model F.

Partitioning - Explicit Mapping-Based Methods



Hardware Partitioning

- Hardware design at the system level is increasingly looking like a platform configuration activity, where most of the features and differentiating aspects, and currently even most of the design and verification costs, are in the software portion of the design.
- This apparently simplifies the design process because software implementation traditionally requires cheaper tools than hardware implementation, and the cost of fixing software bugs is generally considered to be less than that of hardware problems, although the cost of exposing and diagnosing them is not.
- However, it also leads to new requirements in terms of robustness and determinism for the software implementation process.
- In other words, verification can no longer be considered a luxury or an afterthought; formally checking the absence of bugs in the final product is becoming a requirement.

Software Partition

- Nowadays, only timing-critical or power-demanding portions of a system are implemented using hardware components.
- Most of the functions are, in fact, implemented on one or more general-purpose, or semi-custom, or configurable and extensible processor(s) because of the much higher flexibility of the programmable components.
- Embedded processors are becoming so powerful and so many functions are being migrated onto them that the difference between a general purpose desktop PC and some embedded systems, such as cellular phones or MP3 players, is becoming increasingly blurred.
- Embedded systems simply impose more severe constraints, such as smaller memories, lower clock speeds, and real-time deadlines.

Software Partition

- Because processors might execute multiple tasks with different resource and timing requirements, an OS is often needed to manage the software and the available peripherals and communication links.
- These OSs are often very similar to their desktop counterparts, including multiple software layers to abstract the underlying hardware architecture, as in the case of a network protocol stack.
- Therefore, we see that the software architecture is as important, or in some cases even more important, than the hardware architecture.
- Clearly, it requires as much design effort as the hardware architecture, and the impact of poor design on the overall performance of the system will be substantial.
- A well-designed software architecture, on the other hand, allows design teams to efficiently exploit the hardware platform and to easily add or remove features to generate derivative designs starting from a common code base.

Software Partition - Partitioning over Multiple Processors

- Assuming that the hardware platform has more than one processor, the goal is to assign portions of the overall software to the various processors so that some particular metrics are optimized.
- The processors can be either of the same kind, with similar performance and capabilities, or can be very different, often because they are dedicated to different application aspects in the same embedded system (e.g., multimedia and communication).
- In either case, using more than one processor allows one to better exploit the concurrency between the functions.
- However, in the case where two tasks, implemented on different processors, need to exchange a significant amount of data, higher communication and synchronization overhead is imposed.
- This can be ameliorated using application-specific communication mechanisms, such as dedicated communication channels between processors.

Software Partition - Partitioning over Multiple Tasks

- Even if the hardware architecture contains only a single processor, the designer still has the ability to divide software functions among multiple tasks, processes, or threads.
- Scheduling, context switching, and communication overheads can be significant, depending on the techniques that are used to manage access to the single computation resource and the communication links used to connect tasks running on the same processor.

Software Partition - Operating System

- Generally, most embedded systems require some kind of OS to manage a processor or several processors efficiently.
- An OS, another instance of the platform discussed previously, virtualizes a processor (or a set of processors) to a set of tasks, and arbitrates the access to shared hardware resources available on that processor.
- It also provides a means to allocate memory, implement communication among the tasks, and protect a software task from another failing task.

Software Partition - Operating System

- Software partitioning should take into account the presence and the characteristics of the OS because some decisions may depend on it.
- Moreover, it might also be the case that some of the functions or properties of the abstract design can be mapped directly onto functions already provided by the OS, such as drivers for peripherals that are present in the hardware architecture.
- System implementation in this case is easier, but changing the OS will probably require a completely new mapping.

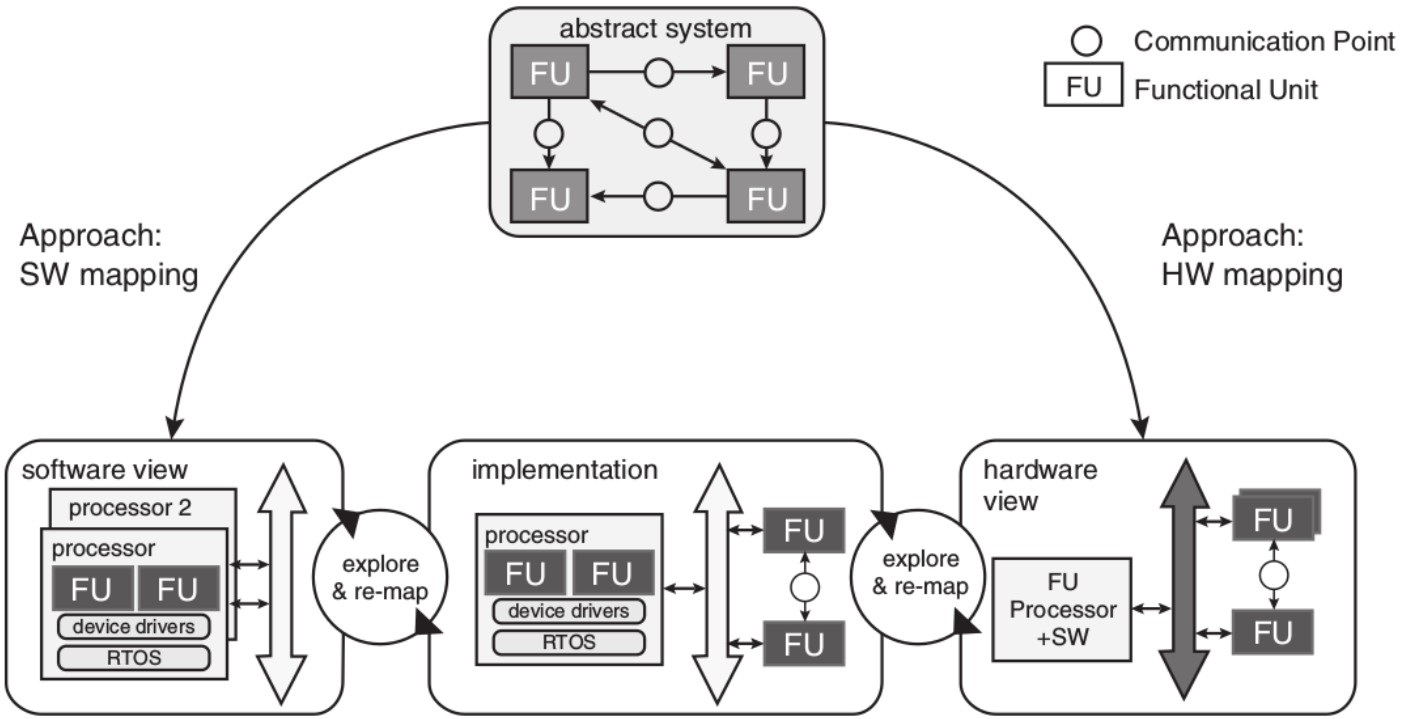
Partitioning - Provocative Thoughts

- So far, engineers have taken a focused approach to the partitioning process, depending on their background.
- A hardware-biased engineer would start to assign the function tasks to an architecture populated with RTL components, as many functions as possible would be assigned to hardware, controlled by a processor and a small amount of software running on that single processor or CPU.
- Simulation against the specification constraints may demonstrate quite quickly that the design has to be extended in the software part by adding an RTOS, for example, to the processor unit.
- Finally, the assignment often has to be iterated before all constraints are fulfilled, resulting in an architecture and associated HW-SW map.

Partitioning - Provocative Thoughts

- In contrast, an experienced software engineer would try to assign as many tasks as possible to software and processors, thus choosing a different architecture as a starting point.
- Even though both approaches have good reasons to exist, and are well-supported by both tools and methodologies, both are also painful: they require numerous iterations of the mapping procedure before finding a solution meeting all constraints.
- However, the final implementation architecture will differ only slightly, depending on individual preferences.

Partitioning - Provocative Thoughts



The Prescription

- 1. The notion of platform-based design is central to the partitioning process, and almost all ESL-based designs are iterations of existing platforms in one respect or another.
- 2. Increasingly, partitioning is moving from a hardware-centric or hardware- biased approach to a software-centric approach, in which functionality is implemented on unique new hardware only as a last resort.
- 3. Mapping alternatives and the availability of good models control the amount of design space exploration that is possible.
- 4. New formalizations for mapping, synchronization, and interface synthesis may reduce the complexity of today's partitioning and exploration process. In the meantime, this job should be reserved for the system architectural team because it essentially controls the quality of results obtainable downstream.