Post-Partitioning Analysis and Debug

- In this phase both hardware and software models that need to be executed together.
- There are also functional and architectural models-the former often expressed as modified executable specification models that may represent functions that have been mapped to hardware or software implementations; the latter often representing hardware block behaviors.
- When the hardware block is a processor, the architectural model is usually some kind of instruction set simulator. In addition, there is
- A requirement that the models collectively support the desired kind of analysis and debug-too much detail and the analysis will be too slow or too late; not enough detail, and it will not provide the necessary information.

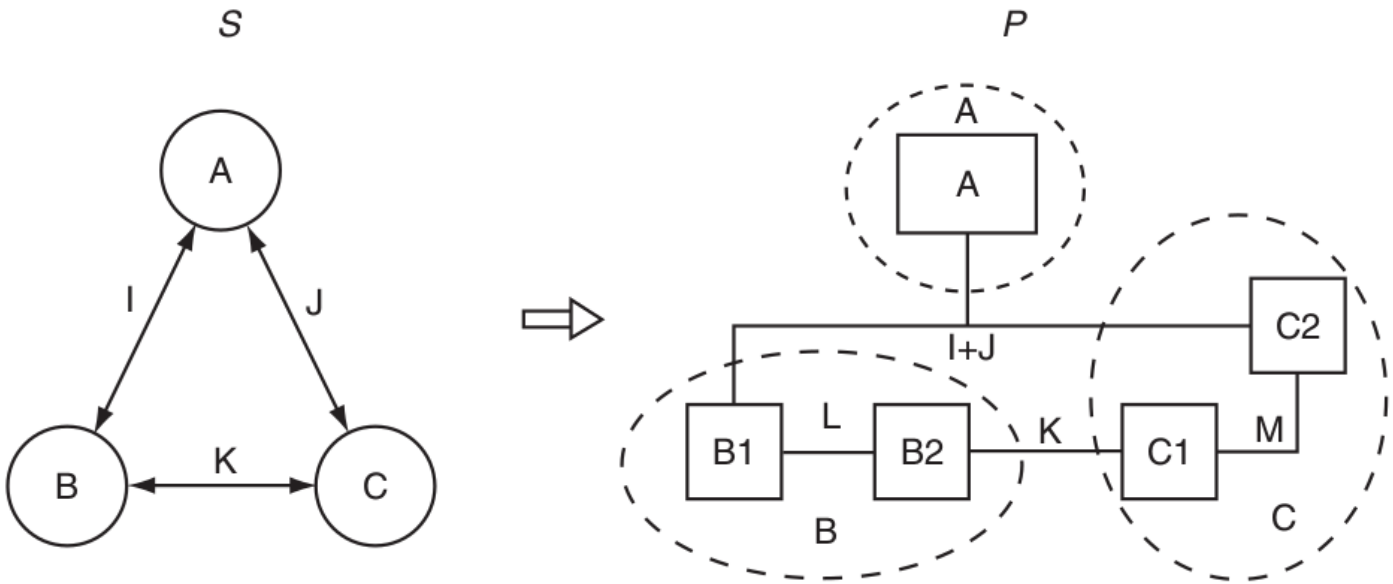
Post-Partitioning Analysis and Debug

- As implementation, verification, and analysis proceed, some of the original design assumptions may prove to be false or inaccurate.
- It may not be possible to implement some function in hardware to meet an original performance constraint or envelope.
- The software implementation on a processor may exceed an estimated or predicted cycle count.
- A configured and extended processor might run a little slower than originally estimated, but the system might still work, despite these implementation realities.
- If the original system was architected and partitioned with sufficient margin specified, then the exigencies of implementation might still fit comfortably within the original system architect's margins of safety or guard bands.

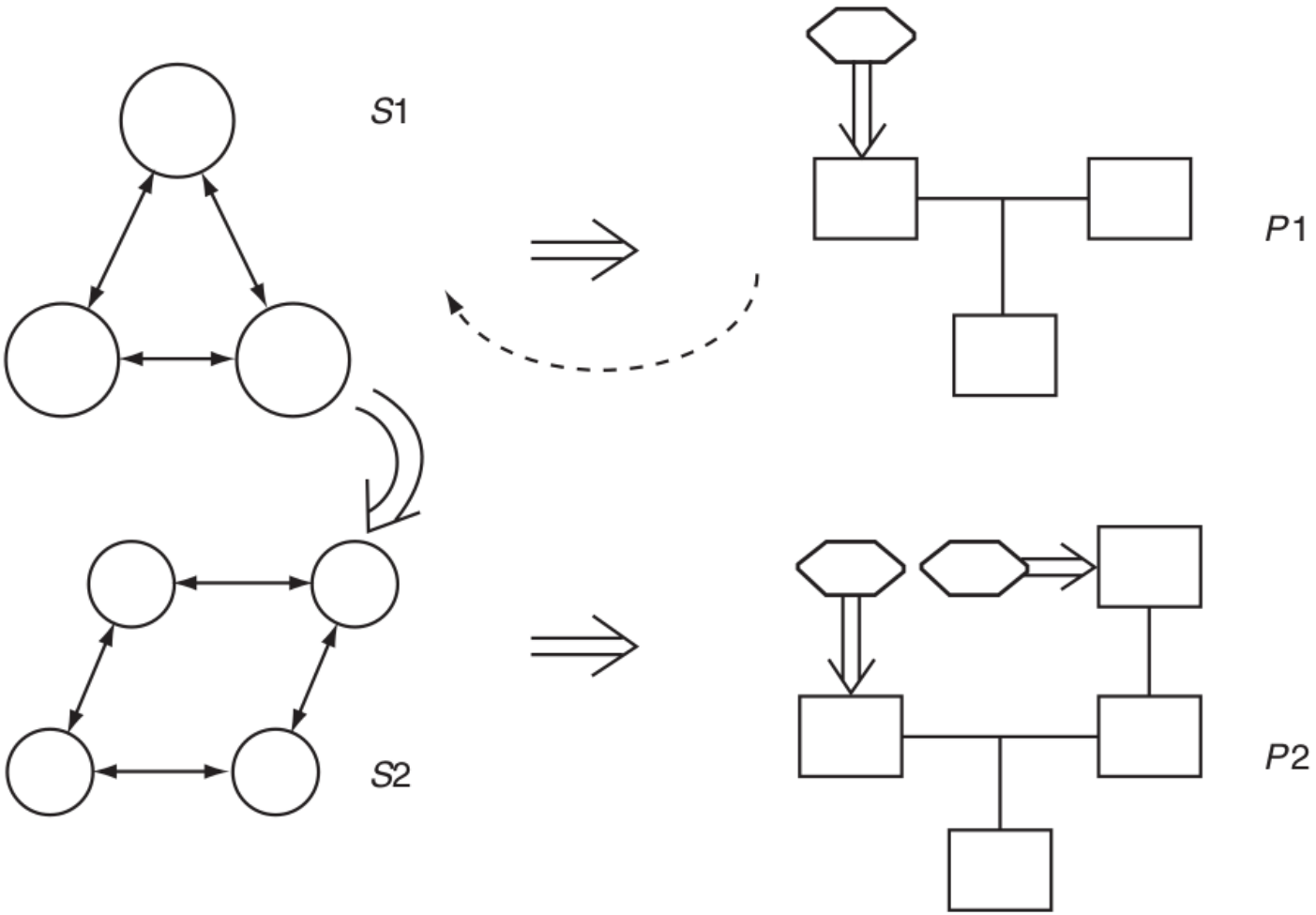
Post-Partitioning Analysis and Debug

- Sensitivity analysis as implementation proceeds, and re-verification with the original system models, annotated with implementation and post-partitioning reality, should be carried out whenever it appears that system constraints may not be achievable.
- In the worst case, the design process will need to iterate back to the pre-partitioning or partitioning stage, or even be abandoned if the design proves unfeasible after all.
- However, it is much more likely, especially if careful analysis and modeling are done early and sufficient design space exploration done during partitioning, that the reality of implementation will merely eat into design margin and not render the design unfeasible.

Post-Partitioning Analysis - Interface Responsibilities



Post-Partitioning Analysis - Model Maintenance



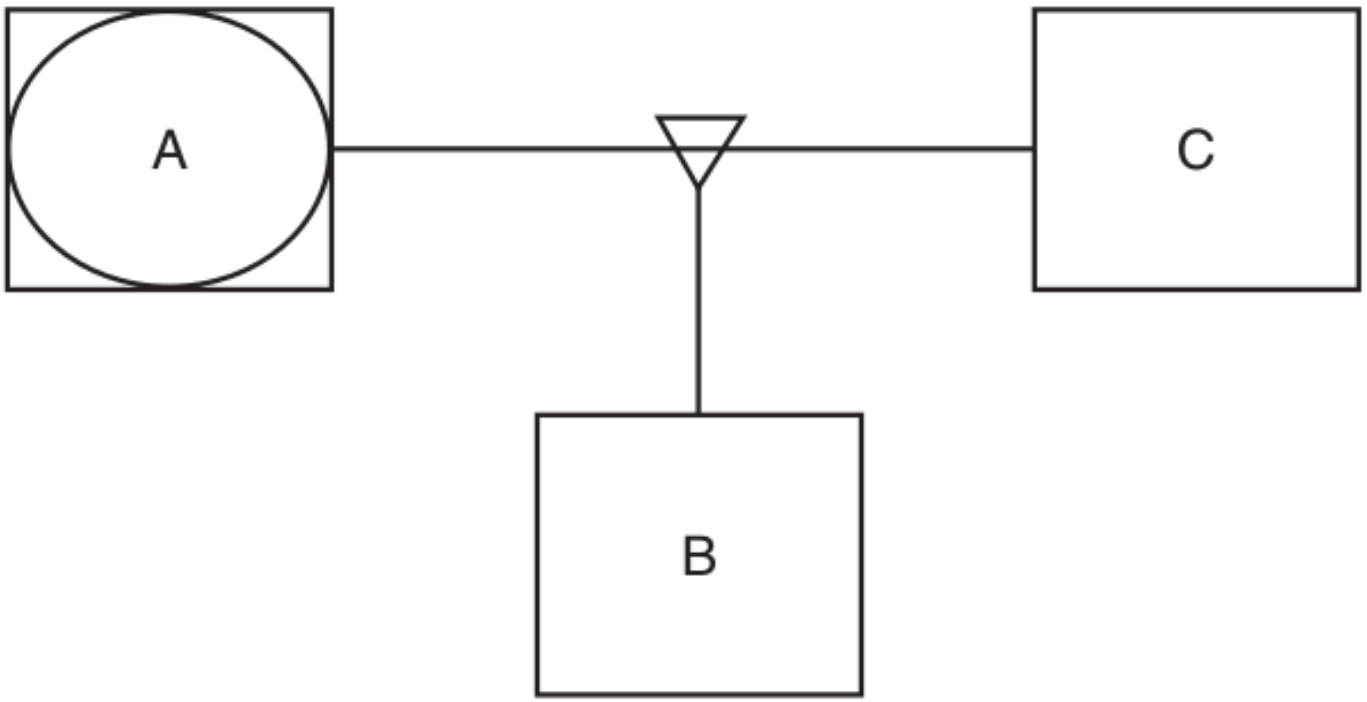
Hardware and Software Modeling and Co-Modeling

- It is important for the models of both hardware and software to be able to execute cooperatively.
- For the hardware and software models to understand each other, they need to communicate in a meaningful way. So although internally they may have quite different modeling paradigms, their interfaces must be compatible in some way.

Hardware and Software Modeling and Co-Modeling

- Modeling options for the software components include the following:
  - Single model - The software can be modeled together with the hardware in a single model, so that the interface relates to the rest of the system-either at a hardware abstraction level or a software level.
  - Filter and translate - Model the software as a software model and use another component to filter and translate the software and hardware communication, depending on the participants.
  - Separata Host Model - Model the software running in combination with the hardware on which it would execute.

Hardware and Software Modeling - Single Model



Hardware and Software Modeling - Single Model - Pros - Cons

- An advantage is that the model need not be defined in fine detail on either the hardware or software side as long as the activity on the interface is accurate enough for the level of abstraction at which the component is being used.
- When the abstraction is at the software level, system models built with this kind of component can be extremely fast, often running natively on the host hardware, and form the basis for virtual system prototypes (VSPs), which can be used to debug application software.
- The problem with this style of modeling is the confusion arising from the melding (rather than separation) of concerns.
- It is difficult to work out how much detail is required, and the partition between software and hardware aspects is often unclear.

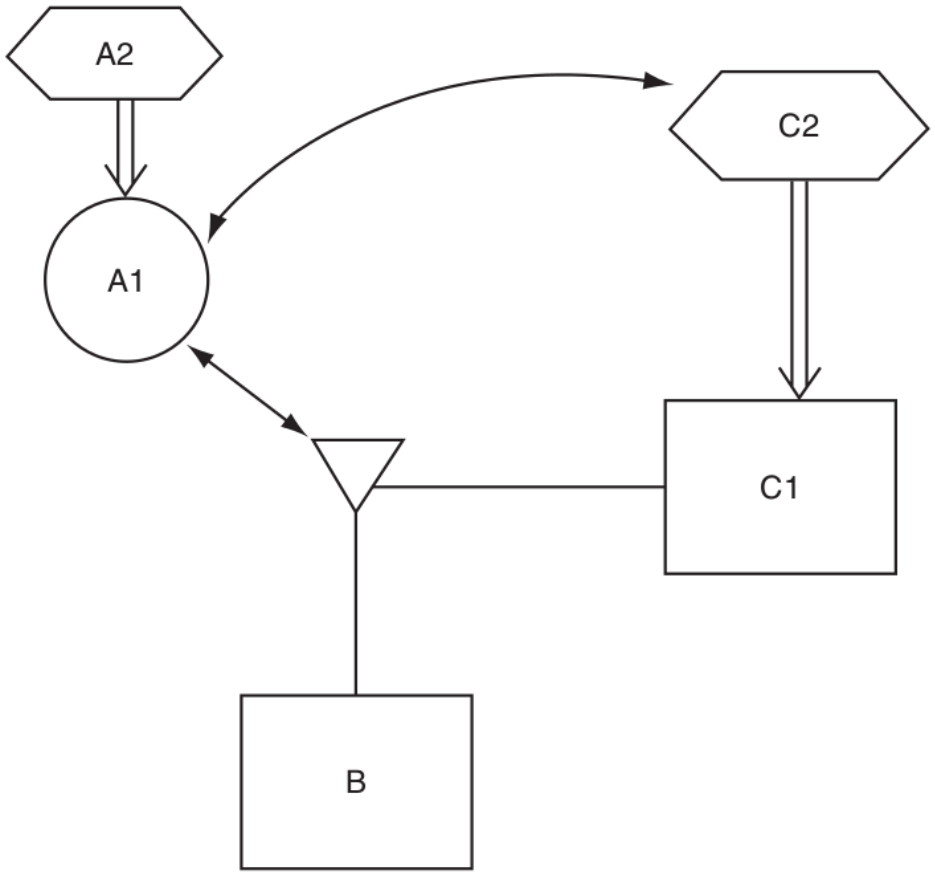
Hardware and Software Modeling - Single Model - Management

- Because of the combination of hardware and software, this sort of model is very difficult to allocate to a hardware or software engineer - a systems engineer is more likely to be the modeler.
- Its use as a vehicle for software validation can be more important, and it may be useful to create or buy a model like this when the design has been stabilized and is less likely to change at the higher levels.

Hardware and Software Modeling - Single Model - Recommendation

- This sort of component is usually used in early pre-partitioned system exploration.
- It can be useful to reuse in post-partitioned system models, but is rarely written specifically for this phase of the design.
- It is more common to create and use it as a very fast VSP for software verification in a separate phase after this design phase, or as an external IP component after the architectural phase is complete and the hardware platform chosen.

Hardware and Software Modeling - Filter and Translate



Hardware and Software Modeling - Filter and Translate - Pros - Cons

- Pro - The model is clearly a software component, so there is a clear separation of concerns
- Con - The software activity that affects the hardware needs to be identified and clearly specified.
- This is not always an explicit communication with other hardware or software components. For example, if a variable is mapped to shared memory, then changes to that memory may affect hardware behavior.

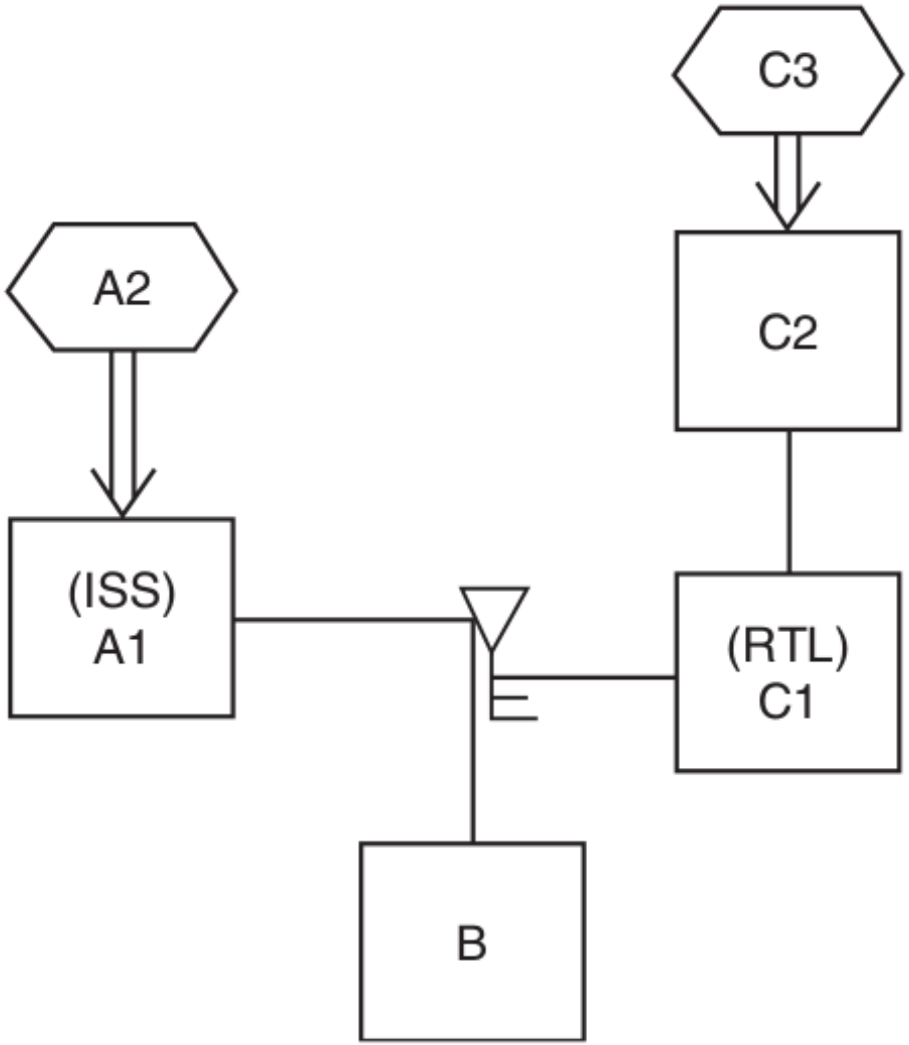
Hardware and Software Modeling - Filter and Translate - Management

- Because there is a clear separation of concerns for the main component, responsibility is easy to assign.
- However, the specification of the activity that is filtered and conducted to the hardware is often done by the software engineers, even though the filter component itself is usually written by the hardware team.

Hardware and Software Modeling - Filter and Translate - Recommendation

- Because most software is not aware of its effect on hardware, this model is often used either for low-level software or when the mapping of the software state to hardware registers and bus activity is available to the interfacing component.
- This style of model can be also useful when the software component is known early but its hardware platform is still malleable.

Hardware and Software Modeling - Separate Host Model



Hardware and Software Modeling - Separate Host Model - Pros - Cons

- Pro - This model can be used at many different levels of abstraction for both the hardware and the software.
- For example, at a low level, it could be running the target machine code on an RTL model of the processor; at a higher level, it could be running the code on an instruction set, cycle-approximate, or cycle-accurate model of the processor hardware.
- This makes it a very flexible style of co-modeling.
- Con - It is very easy to consider this style only for a very low level of abstraction because that is effectively the implementation level.
- However, when created this way, this style of model can be very slow.

Hardware and Software Modeling - Separate Host Model - Management

- There is again a clear separation of concerns for the two components needed.
- The interface required between the hardware and software is needed for the implementation, so using this style would also validate and verify that specification.
- The specification, and thus the responsibility, for this interface should be with the system design team.

Hardware and Software Modeling - Separate Host Model - Recommendation

- The very-high-level models in this style, in which a high-level software model directs a hardware component on the required communication, are arguably similar to the filtering idea discussed earlier rather than a simple hosting.
- So, a transition from that style to this style - as well as to lower levels in this style - can be made smoothly, provided there are interfaces that translate between the different levels of abstraction so that one component can be simply replaced by another.

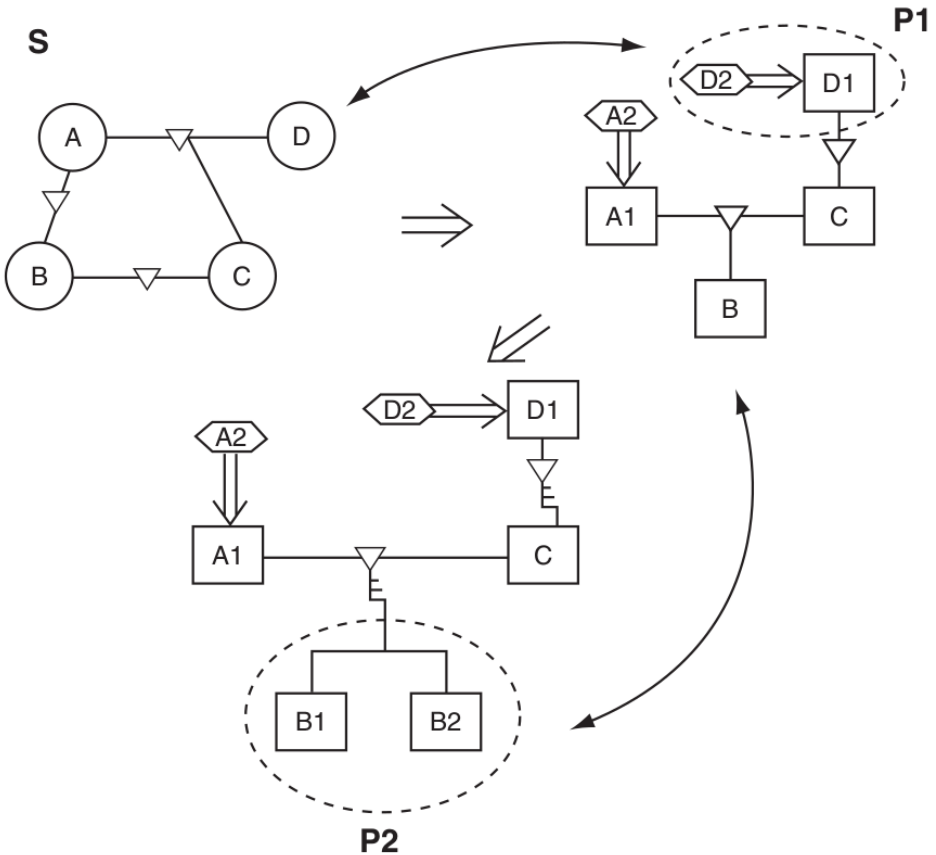
Pre-Partitioned Model Components

- As stated earlier, a truly smooth transition from an abstract system model to a partitioned model of hardware and software components can be performed only if the modeling styles of both pre-partitioned and allocated components are compatible - that is, they can be coherently modeled together, perhaps with some translation.
- A truly useful system model would be able to mix component models from pre-partition phases as well as post-partition phases as the system is stressed and analyzed in order to make the remaining allocation decisions.

Pre-Partitioned Model Components

- Obviously, the pre-partition models would not have the level of detail needed for a complete analysis, but a lot of useful design exploration and defect hunting takes place with these mixed models.
- The key to enabling these component models to interoperate is the clear separation of internal functionality and external communication for each model.
- If this is clear in each model, then the communication can be refined independently or translated so that the component can communicate with another component at a different level of abstraction (or using a different model of computation).

Pre-Partitioned Model Components



Dynamic and Static Analyses

- There are many possible simulation analyses that can be performed at this point in the design route.
- This is the first phase in which some realistic analysis can be done because some important partitioning and allocation decisions have been made-and these may need to be optimized.
- Some of the possible analyses are wholly dynamic, depending only on the simulation of the system models.
- Other analyses are static, depending on the formal analysis of the interface and functional specifications.
- However, a large set of analyses are hybrid analyses, depending on either the static analysis of data generated from a dynamic simulation or, more rarely, a dynamic simulation of a submodel produced from a static analysis.

Dynamic and Static Analyses - Functional Analysis

- Although it may seem that functional analysis is a part of validation and verification, there are cases where it is not always clear whether a model is functionally correct because there may be degrees of "correctness."
- For example, a simulation run in the post-partition phase, with implemented algorithms in the components and real data, is needed to see if the actual behavior of the model matches expectations.
- Often, storage locations (e.g., queues, FIFOs) become full and other parts of the model are delayed waiting for space; or perhaps a location is often empty and a piece of functionality has to wait for data to arrive.

Dynamic and Static Analyses - Functional Analyses

- This kind of analysis gathers data about queue sizes and consequent delays throughout a realistic simulation run.
- The data will be post-processed to see where the bottlenecks are occurring.
- As a result of such analysis, the sizes of storage locations and the performance requirements of the sources and drains to such locations can be checked and optimized for safety or speed.
- Sometimes, such an analysis may lead to a re-partition

Dynamic and Static Analyses - Performance Analysis

- The simulation of a model at the post-partition level with realistic temporal performance annotations allows the gross pre-partition performance requirements to be checked.
- Now that some allocation of components to hardware and software has been attempted, and some idea of concurrency and resource sharing is being explored, it is possible to check performance requirements to see if they are met, or explore to see if they are feasible.

Dynamic and Static Analyses - Performance Analyses

- In addition, the functionality and hence timing of many components depends on the data being processed, so using real data at this phase gives better performance estimates.
- This analysis has a dynamic data generation phase with post-process static anal- ysis.
- However, performance properties can also be continually checked during the dynamic simulation. Many teams leave some of the performance properties switched on for later phases to keep monitoring gross performance parameters in order to identify any problems.

Dynamic and Static Analyses - Interface Analyses

- At this level, communication between the components is often at an abstract level and not with wires or signals.
- For components connected using standard buses, the communication abstractions should be well-tried and tested and would cause no problems when implemented at wire level.
- For newly designed interfaces, however, there is no guarantee that the communication abstractions can be implemented with the predicted throughput.
- By the time the RTL is implemented, it is too late to discover that you cannot implement the interface with the right performance, so it is preferable to spend a little time at this stage to determine if new interfaces can even be implemented.

Dynamic and Static Analyses - Interface Analyses

- If the interface component is refined down to wire level and bidirectional abstraction translation performed between the abstract communication and the wire level, then running a simulation in this mode would check that the intended protocol implementation at wire level would satisfy the protocol at the abstract level.
- In general, this is a good idea for every new protocol.
- They should be designed and verified down to the lowest level before being used.
- If the translation is done manually, this analysis is dynamic and the translation mechanisms in both directions will need verifying as well.

Dynamic and Static Analyses - Power Analyses

- Again, because this is the first phase in which the model resembles a physical implementation, an analysis of activity and movement of data through the system would allow prediction of which components would be most active and which least active.
- This would be mapped to a graph where the power consumed is allocated to the components depending on their activity or known characteristics.
- Although the power consumption estimate would not be very accurate, it would give some gross idea of where problems could lie in the implementation.
- If the interface specifications are known down to wire level, then the activity on each interface would give a good measure of the power that would be expended for communication purposes.

Dynamic and Static Analyses - Power Analyses

- This analysis requires a lot of data from a dynamic simulation, but, even more, it requires a lot of experience from the person performing the analysis.
- The dynamic simulation runs chosen should not just be typical runs but must also include edge cases where activity is concentrated in different parts of the functionality.
- Previous experience or real data in mapping the chosen activity metrics to power in a real implementation is extremely useful.

Dynamic and Static Analyses - Area Analyses

- Having partitioned the system down to a level where the hardware and software parts are separate, the complexity of the hardware components and the interfaces between them should give a good indication of the probable area of the final implementation.
- Again, an estimate using no previous data will usually have a large margin of error.
- It is best to use standard complexity metrics for function and interfaces and use previous experience and data.
- If a new metric is being used, then post-analysis of a previous project to get some idea of the scaling factors is also a good idea.

Dynamic and Static Analyses - Area Analyses

- A lot of knowledge and experience is gained in every project-successes and failures-that should be captured so that it can be used in the next project.
- It is no coincidence that the most successful companies doing systems design are those that have some continuity between projects and don't just start afresh for each one, pass- ing no knowledge between them.
- These companies also often perform a postmortem on a project after its completion to see what worked well and where improvements can be made.

Dynamic and Static Analyses - Cost Analyses

- Cost analysis uses the most heuristics of the different analyses that can be performed because the cost depends not only on the choice of components, platforms, and technology, but potentially on the complexity of the custom parts to be designed.
- The costs associated with the choice of partition-components, platforms, and technology-and support and lifetime costs are usually straightforward to work out if the unit costs have been captured somewhere.
- Estimating design complexity and its effect on cost is probably best done by using software complexity metrics on the model components, along with data captured from completed projects. An extrapolation from complexity metrics without any guidance from real projects will almost certainly have a very large margin of error.

Dynamic and Static Analyses - Debug Capabilities Analyses

- When a complex system is put onto a chip, the only points that can be observed are the external pins of the device.
- Given that over one half of all chips have functional errors in them the first time they go to silicon, it is important at this stage to decide on the capabilities that are to be put into the system to enable a bug or unexpected behavior in the chip to be analyzed and debugged.
- Most of these capabilities require some additional logic to be added into the components that must be integrated and coordinated by a central capability and the results of that fed to the external pins of the device, often being integrated with the test capabilities of the chip.
- Although test logic is normally inserted in the back end of the implementation process, it is unwise to consider the debug systems that late in the process and this in turn makes a strong argument for moving DFT up earlier in the design flow.

Post-Partitioning Analyses - The Prescription

- Within a project, division, or organization, a standard set of model abstractions must be defined. The uses for these models should be made clear.
- The interfaces in a system should have clear responsibilities and ownership.
- Interface models should bridge all of the defined abstraction levels so that hybrid models can be created quickly and easily.
- Build knowledge over time and ensure that past experiences are used to construct better processes, models, or characterization data.
- Do not expect chips to work completely the first time. Plan how you will diagnose and fix problems early.