Pre-Partitioning Analysis

- Pre-partitioning analysis refines the system specification models and defines additional or elaborated design constraints derived from the high-level requirements, before the main partitioning of the design into hardware and software; into hardware blocks and software functions; and, in the case of multiprocessor systems, into a specification of multiple processors and their interconnection.
- However, the division of the ESL design process into rigid pre-partitioning and partitioning phases, although allowing easy explanation of a design flow, is not very representative of real design.
- First, a product design may be based on a platform, where part of the design is pre-specified and pre-implemented and validated, and the new (or derivative) design adds new or modified functionality. In this case, of course, many of the partitioning decisions are already made.

Pre-Partitioning Analysis

- Second, for any design, it is quite unrealistic to prevent all implementation-level decisions, including partitioning ones, to creep into the mental design process while analyzing the specifications.
- Although static analysis methods are useful, dynamic analysis based on executable models and specifications is far superior in its ability to give the design team a firmer grasp on specification details and design constraints, so every attempt should be made to ensure that at least part of the specification consists of executable models.

Pre-Partitioning Analysis

- Depending on style of the specification, there are several ways in which analysis can be done:
  - Static Analysis
  - Platform-Based Analysis
  - Dynamic Analysis
  - Algorithmic Analysis

Static Analysis

- If specifications and models are not executable, then analysis must be based on static methods-usually informal-that will allow us to analyze the specifications and derive interesting properties or constraints of the resulting systems.
- Over the years, a number of methods have been developed for informal static analysis of systems, and these can still be applied today even though the focus has shifted to executable models.
- It is not so important that the absolute error for static analysis, based on estimators derived either theoretically or from historical experience, be low, as that the tradeoff decisions made based on the estimators be monotonic.

Static Analysis

- If analysis of choices A and B indicates that A is preferred to B, then the actual system design results for the criteria under analysis match this (A will prove to be better than B, within some reasonable envelope of uncertainty).
- However, even in the absence of a strong historical basis for the underlying predictor models, simple analytical models have proven to be of value in making decisions on lines of development.

Static Analysis

- Static analysis examples are:
  - Function Point Analysis
  - Analysis of Hardware and Hardware-Dominated System Specifications
  - Traditional “ility” Analysis of Systems
  - New languages - Rosetta, SysML

Static Analysis - Function Point Analysis

- Many years ago, in the software community, there was a considerable emphasis on deriving metrics that would allow prediction of characteristics of software, based on more sophisticated measures than, for example, "lines of code".
- Much of this was motivated by a desire to predict the development time and effort required for complex software projects.
- A specification was used to derive a decomposed function model, where a function model was a "transformation of an incoming stream of data into an outgoing stream of data." This function model was then complemented by a data model, which represented the retained state of the system.
- The final specification model was a behavioral state transition model.
- Then, metrics applied to the function, data, and behavioral state transition models would be used to derive a total "weight" for the system development project

Static Analysis - Rosetta

- Work with aspects of systems
- Functional aspect of register

```
facet regFcn
    (x::input word(4); z::output word(4);
    rst,le,clk::input bit)::discrete_time is
    s :: word(4);
begin
    sup: s' = if reset=1 then b"0000" else
        if clk=1 and event(clk) and le=1
            then x
        else s
        end if;
    end if;
    zup: z = s;
end facet regFcn;
```

Static Analysis - Rosetta

- Power aspect of registrer.

```
facet regPwr
    (rst,le,clk::input bit;
    switch,leakage::design real)::state_based is
    export power;
    power :: real;
begin
    pup: power'=power+if clk=1 and event(clk)
        then if le=1 then switch
        else leakage
        end if;
    else leakage
    end if;
end facet regPwr;
```

Platform Based Analysis

- An integration oriented design approach emphasising systematic reuse, for developing complex products based upon platforms and compatible hardware and software virtual components, intended to reduce development risks, costs and time to market.
- Although a platform may be fully delineated with platform architectural mod- els, IP models, and functional models of software components and other elements, any derivative design based on a platform will have new functionality that must be implemented.
- The difficulty that platform-based design primarily poses to the design team is the strong risk that they will be biased unnecessarily toward the partitioning decisions already made in the platform.
- There is no technological fix for existing platform bias in the pre-partitioning analysis phase. There is only design discipline. In this case, each new requirement and associated specification and models should be analyzed as if one were starting with a clean slate for the design.

Dynamic Analysis

- The main evaluation criteria narrow to some extent when we turn to dynamic execution or simulation of the specifications by executable models.
- Here we look to characteristics that depend on simulation, such as more accurate performance estimation (delay, throughput, latency) and factors that have an impact on time-based performance, such as the causes of congestion, arbitration policies and priorities, and scheduling algorithms for execution of models.
- From the dynamic simulation of executable models at the pre-partitioning stage, we need to derive important characteristics of the underlying functionality that we will use in partitioning. These include:
  - Computation burden
  - Communication burden
  - Power burden

Dynamic Analysis

- If a specification model for a particular function is in an executable form in C or C++ , but does not satisfy the constraints of any particular toolset and associated model of computation, as in an algorithmic analysis tool, then host-based execution and associated analysis is the current state-of-the-art approach for dynamic analysis.
- Software profiling tools can be used to produce estimates of computational burden when running many test sets and test scenarios.
- By weighting instructions with estimates of power consumption per instruction, the power burden can be estimated, assuming one has power consumption figures for memory accesses.

Dynamic Analysis

- Adding simple monitoring functions can be useful to track the amount of data and control communications that executing the functional model entails.
- It is of course important to remember that the functional model is at best an imperfect analog of the real implementation.
- Thus, all estimates derived from either static or dynamic analysis of the functional models that make up a system specification are not necessarily totally accurate, and should be used with care in making decisions.
- Monotonicity of estimates may be useful in making decisions despite inaccuracies.

Algorithmic Analysis

- Algorithmic analysis for such evaluation criteria as estimated execution or computation load, amount of data transport required for processing frames, packets, or tokens through the algorithm, error rate, bit error rate, and relative computational balancing, is important, especially for dataflow algorithms used in signal and image processing applications.
- The process of floating-point algorithm to fixed-point conversion is also important during this phase of system design and, if possible, optimizing the fixed-point specification is useful before final partitioning and implementation.
- More optimization is possible post-partitioning, when the characteristics of the final implementation platform are better known.
- Nevertheless, initial setting of fixed-point range and precision requirements in the pre-partitioning stage will result in less work post-partitioning.

Analysis Scenarios and Modeling

- Filter Design Example

Downstream Use of Analysis Results

- The case studies discussed earlier in the context of dynamic algorithmic analysis illustrated the possibilities in using the models, specifications, and analysis out- puts in downstream implementation and verification.
- Among the possible downstream uses are:
  - Floating-point and fixed-point models and results used as golden verification environments for actual implementations in both hardware and software
  - Algorithmic specifications that can drive software code generation for execution on target processors and DSPs
  - Algorithmic specifications that can drive hardware code generation for RTL level synthesis
  - Co-simulation between system-level simulation, RTL simulation, and software execution of code on instruction set simulators

Downstream Use of Analysis Results

- HLS is an interesting target for downstream use of executable specification models from this stage of the process.
- A number of experiments and attempts have been made to use MATLAB and Simulink to drive hardware synthesis.
- Software code generation represents a downstream use goal for specifications that is more talked about than used in practice, although the UML community has long held this up as one of its goals.

Summary and Future

- The greatest success of pre-partitioning techniques is in specific models of computation and associated tools, such as dataflow analysis for signal and image processing applications.
- This leads to a thought that perhaps the future of ESL will see a distinct specification -> analysis -> partitioning -> implementation -> verification flow replaced by a number of domain-specific integrated flows, each dealing with the particular subsystems of an integrated product that fit into those design domains.
- If generalized control design could be done as successfully as signal processing design in wireless and wired communications; if software implementations could be generated from abstract models in UML or SysML; if HLS takes off and really solves the implementation problem for high-level specifications, then it may be much harder to distinguish specific or distinct phases in the ESL flow. Instead, we could end up with the way to do dataflow algorithms, the way to do control algorithms, the way to implement software, and so forth.

The Prescription

- Because a specification requires a lot of effort to create, it must be analyzed and maximum value derived from it through pre-partitioning analysis.
- At the same time, one must avoid both "paralysis by analysis" and "death by analysis." One should not assume that the value of pre-partitioning analysis will justify an infinite amount of time on the analysis phase. Rather, specific questions should be listed and answered during the analysis.
- Gaining some idea of functional or algorithmic characteristics through dynamic simulation of any part of the specification that is executable is worth doing.

The Prescription

- For portions of a system specification that fit into a particular design approach and satisfy the requirements of a particular model of computation (such as dataflow algorithms), the use of any available ESL tools should be investigated and adopted.
- Design groups should evaluate new specification notations and analysis methods and tools as they emerge. Keep a watch on new academic methods.