ESL

- Electronic System Level
- Electronic System Level design, or "ESL," is an emerging electronic design methodology which focuses on the higher abstraction level concerns first and foremost.
- ESL seems to be a collection of many different activities and methodologies for designing "systems" of various types. These "systems" are electronic based, yet they involve both hardware and software.
- ESL is about system, abstraction, and process.
- People often informally think of ESL as being the design of systems including a mixture of hardware and software.

Model

- A model is a pattern, plan, representation (especially in miniature), or description designed to show the main object or workings of an object, system, or concept.
- Models allow us to reason about things that do not exist yet, are intangible, or are not fully understood. A model helps us to understand something. That understanding may allow us to refine the model or make decisions based on our expectation of the fidelity and accuracy of the model.

Example - Models of Universe

- Earth was at the center of the Universe and that everything rotated around it.
- Copernicus' heliocentric model - Newton theory
- Albert Einstein - General Theory of Relativity
- The simplest model that gets the right numbers should be used.
- Compare this:

$$F_g = \frac{G m_1 m_2}{r^2}$$

$$R_{\mu\upsilon} - \frac{1}{2} g_{\mu\upsilon} R + g_{\mu\upsilon} \Lambda = \frac{8\pi G}{C^4} T\mu\upsilon$$

Importance of models

- Models are a fundamental aspect of everything we do in the EDA industry.
- Models can abstract away the physics that makes transistors operate, they allow us to verify a design before it has been built, they allow us to increase design efficiency by having tools that can transform one model into another model that is closer to the final implementation.
- The semiconductor development process is about the creation, application, and transformation of models.
- Models are expensive to create and maintain, so we must limit the number of them that get used in the development process, often resulting in models being used for things that they were never intended for.

Importance of models

- If the model contains too little information it will result in inaccurate results
- Having too much information is equally bad. While we may then get very accurate results, the processing time necessary to get that result is not acceptable.
- Engineers are thus continually playing a trade-off game between the time and expense of creating models that would enable them to perform a specific task and the value that those models would provide.

Types of Model

- There are many types of models, including conceptual, physical, logical, mathematical, statistical, and visual.
- A conceptual model is a qualitative model that helps to highlight important connections between processes. Conceptual models exist primarily in the problem domain and not in the solution domain. Conceptual models are often process flow diagrams, Petri Nets, event graphs, or some aspects of the Unified Modeling Language.
- A physical model is most often something that can be created, observed, and manipulated. One or more aspects of the model have characteristics that are similar to the real system and thus allow concepts to be analyzed and understood.

Types of Model

- A mathematical model enables equations to be analyzed. In our industry most of these will be numerical models although occasionally analytical models may exist as well. Statistical models allow variances to be modeled and to find means and variances.
- Visual models help us to visualize how something will operate within the real world by providing some graphic or visual outputs, often with animations.
- A logical model includes some aspects of the structure or form of the device that is to be built. This includes architectural models, transaction level models, and any model that includes time. These are thus implementation models and help us refine the ideas into a complete solution.

Models of Computation

- A model of computation is an attempt to distil the fundamental aspects of a system's ability to solve problems.
- A model of computation abstracts away many of the details of the machine so that it can be both fast and accurate in terms of the analysis which is to be performed. An example of this is the way in which various models of computation treat time.
- There is no one abstraction of a system and thus no one model of computation that is useful for all tasks in a typical ESL flow. Multiple models of computation will be required to progress from an idea to an implementation.
- The most fundamental model of computation in the software world is the von Neumann architecture.
- For the gate and RT levels of abstraction, the predominant model on which all languages have been based is the discrete event model.

Model of Computation

- Languages are ways to map the capabilities of a model of computation into something that can be described and communicated.
- It is often thought that the language is the syntax and the model of computation underlying it is the semantics.
- There are languages that have been constructed that do not have a complete model of computation.
- Languages such as C encode the limitations of von Neumann architecture.
- It is clear that we need different models of computation to describe hardware systems and software systems even though they share many of the same needs, such as a way to express concurrency and time.

Simplification - Abstraction

- Models allow us to simplify a problem, often down to the essential elements necessary to perform a particular function, computation, or type of analysis. The primary means of simplification are abstraction, elimination, and structure.
- In the software realm, the processor directly implies the lowest level of software abstraction, that is, the instruction. The processor takes care of everything below the instruction level.
- A language like C adds another layer of abstraction, in that each statement written in C will result in several instructions being created when passed through a compiler.
- In the hardware world, there are two primary means of abstraction. These are time and data.

Simplification - Structure

- While a final chip is a flat structure, models can represent this in a hierarchy that enables us to group certain things together, organize by function, perform duplication, and divide a problem.
- If you tried to describe a system in a fully flattened form, very few tools would be able to handle the size and complexity of the resulting system. Hierarchy is very necessary and helpful.

Models and Languages

- A modeling language is any artificial language that can be used to express a set of concepts in a consistent way such that independent tools can be built upon that language and can in most cases lead to exactly the same interpretation of the model.
- Languages are basically built from three sets of concepts, namely semantics, abstract syntax, and a concrete syntax.
- We have already discussed semantics, especially in the context of the models of computation. The abstract syntax is used to describe in a language neutral way how components are connected structurally, but says nothing about the semantics of those connections. And finally the lan- guage syntax itself defines the keywords, how are they formed, the detailed rules for expressions, etc.

Models and Languages

- Most programming languages in use today are imperative languages. This is a class of languages in which statements are executed in order and modify the state of the system. The dominance of imperative languages should come as no surprise because computers themselves are imperative machines.
- While imperative languages describe a set of ordered statements that are to be executed, declarative languages state what needs to be accomplished, but not the way in which it will get done.
- Emerging use for declarative languages comes from the formal side of the verification process: PSL and SVA.

Models and Languages

- Functional languages are those languages that we use to define the function of a system. These are thus the most commonly used languages of our industry.
- Both imperative and declarative languages can be used to define functionality.
- The dynamic languages and models can be further subdivided into algorithmic approaches and architectural modeling approaches. The most prominent and widely used architectural modeling language is SystemC.
- Algorithmic modeling languages can be subdivided into mathematical modeling textual languages, the most famous being MATLAB and dataflow modeling languages, like Simulink.

Models and Languages

- Non-functional languages and models are intended to capture characteristics of a system such as constraints (e.g., on energy consumption, area, cost) that define boundaries on the allowable or feasible implementations of those systems - SDC.
- Meta-models are usually pragmatically defined aggregates of information about a system or its components that are "left out" of all the other models in use - SPIRIT's IP-XACT.

Language Problem in ESL

- ESL spans two domains, namely the hardware and software domains. These two domain use different computation models and that is the problem.
- With the migration to ESL, the discrete event semantic is too limiting.
- ESL needs the language which potential can model both domains equaly good. That language is SystemC.

SystemC

- SystemC is based on a single fixed model of computation, the same as the existing HDLs: namely, it follows the discrete event semantics. But because SystemC is built on an extensible framework, namely C++ it is much more flexible.
- In addition, given that it is based on C++, which itself is an extension of C, it fully supports the legacy software that exists.
- It has been constructed to support multiple levels of abstraction within the discrete event domain ranging from RTL constructs up to transaction modeling.
- It can be extended to support dditional communications capabilities - TLM 2.0

Taxonomy

- A taxonomy is a characterization of objects or concepts based on the relationships that exist between them. The aim is classification.
- A taxonomy can be represented in a hierarchical graph or table of attributes, where each of the attributes identifies a particular element of the differentiation.
- The ESL taxonomy is composed of five main axes.
- The axes are temporal, data, concurrency, communication, and configurability. These axes are not completely orthogonal.

Temporal Axis

- The temporal axis defines the timing in the model.
- The defined points are:
  - Partially ordered events
  - System event
  - Token cycle
  - Instruction cycle
  - Cycle-approximate
  - Cycle-accurate
  - Gate propagation accurate

Temporal Axis

- The most abstract point defined on the scale is partially ordered events. This means that we know when something will start and finish only in terms of its relationship to when other things start and finish in this particular execution. We cannot place any notion of actual times on these events.
- Token cycle accurate is typical in dataflow systems where a data arrival clock can be thought of as a regularly scheduled event.
- The instruction cycle, cycle-approximate, and cycle-accurate points all have the notion of an actual clock with a known period, and thus actual times are known.

Temporal Axis

- At instruction cycle accuracy, no account is usually made of wait states or the impact that hardware may have on the execution times of software. The memory accesses normally happen only in a virtual sense.
- At cycle-approximate accuracy, the actual operations of the bus or memory accesses are present, but the timing between them is not known precisely.
- At the cycle-accurate point on the axis, exact cycle counts are known.
- Gate propagation accuracy is the final point on the axis at which the timing within the clock period is also known precisely.

Data Axis

- The data axis defines the level of precision of data.
- The points on this axis are:
  - Token
  - Property
  - Value
  - Format
  - Bit Logical

Data Axis

- A token indicates that some data is moving through a system, but its size and content are unknown.
- A property may be an enumerated type where a name is given to something, even though how this name will be represented is not defined.
- Value could be an integer, floating-point value, or similar data type, where its numerical accuracy is known but how it is represented in hardware is not known.
- Format is a processor-like data format, dealing with issues such as endianess, and fixed- vs. floating-point.
- Bit logical provides that mapping of value into the hardware that will store it.

Concurrency Axis

- Concurrency defines the amount of processing or execution of an application that can be performed simultaneously.
- An implementation does not have to utilize all of the available concurrency because this may produce a solution that does not fit the nonfunctional requirements of the design, such as cost, size, or power consumption constraints. In addition, more concurrency increases the difficulty of verifying an implementation.
- Points on this axis are:
  - Sequential (none)
  - Multi-application
  - Parallel
  - Multithread
  - Pipeline
  - Signal level

Concurrency Axis

- Sequential (none) — No explicit concurrency exists in the function. This does not mean that it is not possible to extract concurrency from the function by looking at control or data dependencies, but that no indications exist in the function to identify such concurrency. An example of this would be ANSI C descriptions.
- Multi-application — The concurrency that exists here is very coarse grained and not built into the operating characteristics of the system directly. In other words, two independent functions may share some data contained in a file, but one function has no direct influence on the other. An example of this would be a Microsoft Word document and an Excel spreadsheet, where it is possible to embed data or charts from the spreadsheet into the document. Whenever the document is opened, the updated chart or data from the spreadsheet would be imported.

Concurrency Axis

- Parallel — This level of concurrency concerns multiple functions that are operating independently, but may need to cooperate for certain activities. An example is a transaction processing system where data is explicitly shared between each function or instance of the function. Data locking on those accesses has a direct impact on the running of each function and may trigger further functions to be executed. Normally, one would not be able to predict when such synchronization points would happen because the events that initiate them are independent.
- Multithread — A single function may have been defined with multiple threads of execution. This concurrency is thus explicitly built into the operation of the function and may have explicitly defined synchronization points. The means of synchronization and data transfer are built into the function. Examples are: the operating system running on a computer, multiple pieces of a hardware system that have divided up the task into independent functioning blocks.

Concurrency Axis

- Pipeline — In general, pipelining defines the streaming of data from one operation to the next in a controlled manner. This also highlights the hierarchical nature of a system, such that within a coarse pipeline stage, we could consider a finer level of parallelism and communications also implemented as a pipeline. There is a range of different granularities when we talk about pipelines. At the coarsest level, there could be large blocks of computation that could be executed within a dedicated processor, programmable coprocessor, or fixed-function custom hardware block, and the results of this may be fed to another, similar subsystem. Examples of these kinds of pipeline stages could be a Fast Fourier Transform (FFT) or multitap filter. At a finer level, pipelining is very common in processors where the main data path will be constructed as a pipeline, with stages such as instruction fetch, decode, argument fetch, execution, and writeback. It could also be considered to be the default level of definition for an RTL description, where the combinatorial actions to be performed between registers are defined.

Concurrency Axis

- Signal level — This is a very fine level of concurrency that is typical in a gate-level or asynchronous description. Multiple combinatorial paths exist between two synchronization points and signal transitions progress through the system as fast as the circuitry will allow. This may create transient values along the processing chain and would normally be combined with some degree of pipelined design so that synchronization points can be defined.

Communication Axis

- When more than one processing element exists, there must be communication between them. That communication can take many forms and is highly dependent on the architecture of the final solution.
- The points on this axis are:
  - Point-to-point
  - Buffered
  - Coprocessor
  - Memory
  - Bus based (high speed)
  - Bus based (low speed)
  - None

Communications Axis

- Point-to-point — This allows two concurrent functions to communicate with each other directly without any additional form of control between them. This would generally be found only in hardware solutions because it implies the continuous flow of information rather than something that is intermittent, periodic or not.
- Buffered (also includes FIFO) — This is the most elemental form of con- trolled communications that can be made between two concurrent tasks. It allows for the execution rates of two functions to be different and thus pro- vides some degree of electrical and timing isolation between them. If the buffer depth is greater than one, as would be the case with a FIFO, then rate independence can also possibly be assured, at least to the depth of the FIFO.

Communications Axis

- Coprocessor — Although this may sound like a processor-dependent term, it means that the information processed across the two functions is being shared in some manner. It is likely that one of the functions is the owner of the infor- mation and may have to do some work in order to make it available to the other function. In other words, a producer–consumer relationship exists between them. In the case of a processor/coprocessor pair, the processor would own the information, but at the time of the request the information may be in a register, cache, or memory. The coprocessor is unaware of its actual location, but it does affect the rate at which the information is returned. A coprocessor can be programmable or fixed-function hardware.
- Memory — Memory is a very common way to transfer information between two functions. At this level, it is assumed to be multiport such that each func- tion can read or write to the memory independently. It would also imply that they have a dedicated connection to the memory and not have to access it through a bus mechanism (see the next point on the scale). Examples of this often

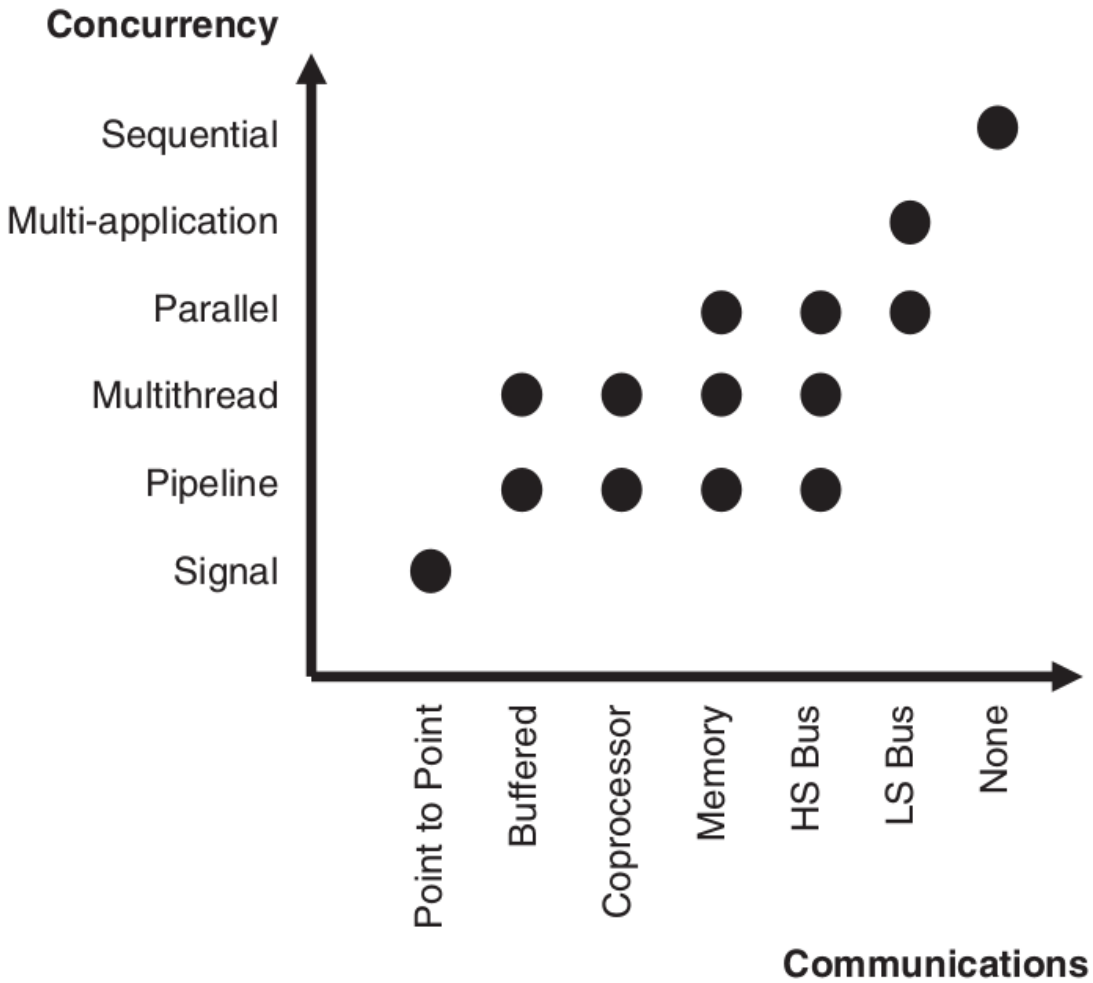
Communications Axis

- Bus based (high speed) — In software systems where the function is run on a processor, almost all communications are made through one or more buses, each of which may have different transfer rates or latencies. Several buses may be used to make the necessary transfer, with bus bridges mapping the requests between them. This category is meant to include all buses directly associated with the processor and does not, for example, make the distinction between the Advanced Microcontroller Bus Architecture (AMBA), Advanced eXtensible Interface (AXI), and AMBA Advanced Peripheral Bus (APB) buses of an ARM processor. This category would also contain the emerging architectures called Network On Chip (NoC). When a communication request is made between any two of the processors, the routing resources within each of the processing elements act as a bridge between each of the processor buses.

Communication Axis

- Bus based (low speed) — In this context, a low-speed bus means that there is a more extensive protocol being used on the bus to ensure transport integrity, or that is capable of allowing things to be connected at much larger distances. Examples of this within a computer world would be a Peripheral Component Interconnect (PCI) bus or a Universal Serial Bus (USB). It could also include an Internet connection where functions would be communicating over great distances and have very long latencies.
- None — If the functions do not need to communicate, then no channel is necessary for their communication. This also implies that the concurrency between the functions does not create any issues with which the system designer must be concerned.

Concurrency and Communications



Configuration Axis

- This axis descibes the configurability of the solution.
- The points on this axis are:
  - Fixed
  - Design
  - Configurable
  - Programable
  - Dynamically reprogrammable

Configuration Axis

- Fixed — The user of a fixed block or device has no ability to make any kind of changes to it. A fixed design may be provided at any level of abstraction, in hardware or software. It may be provided as a black box lacking any kind of visibility into its internals, although this does not have to be the case. A compiled object file or executable is an example of a fixed design in software.
- Design — Configurability at the design level is possible for any design where the original source is available. Almost all designs are configurable at design time. This is considered the normal design flow for hardware and software components.

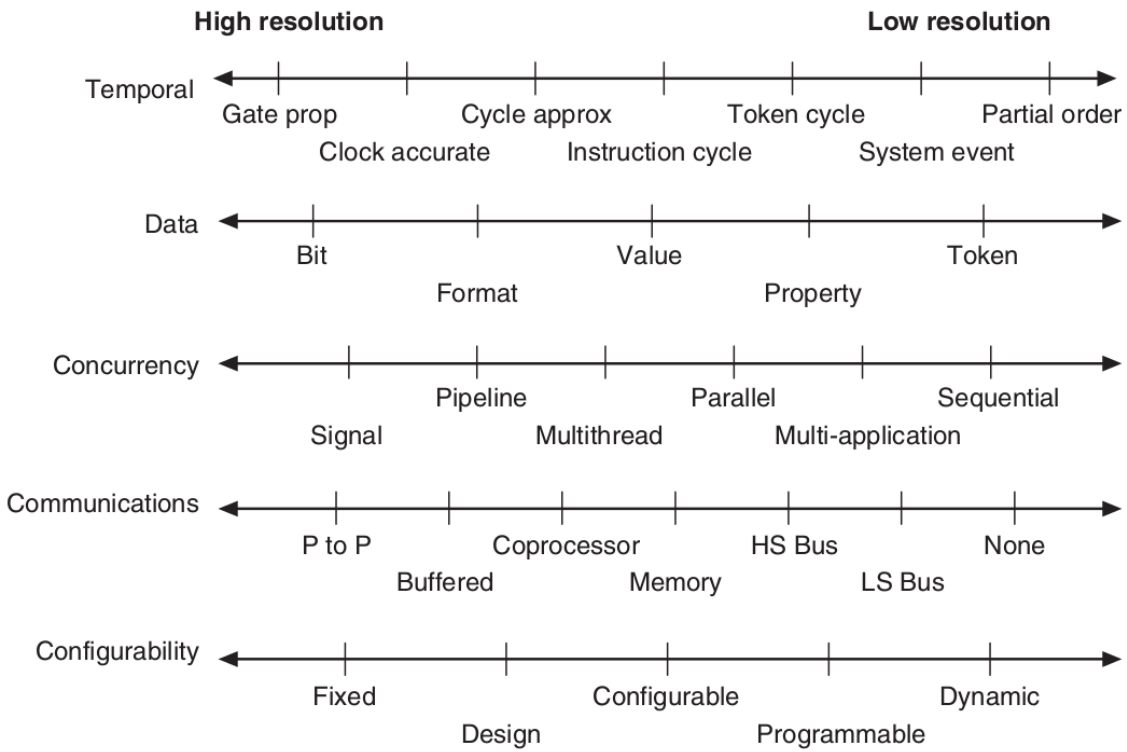
Configuration Axis

- Configurable — Configurable IP blocks come with a set of configuration options. These blocks can be personalized to take on many different roles. Examples would be a Universal Asynchronous Receiver/Transmitter (UART) or a communications processor with options to control the protocol to be used. Once these options are set, they cannot be modified during operation; neither can additional configuration options be added without re-design.
- Programmable — Programmable devices such as FPGAs and PLDs can take on any number of personalities by loading configuration information at start-up time. New configurations can be defined over time. Because of their slow configuration times, they are not considered to be dynamically reprogrammable.

Configuration Axis

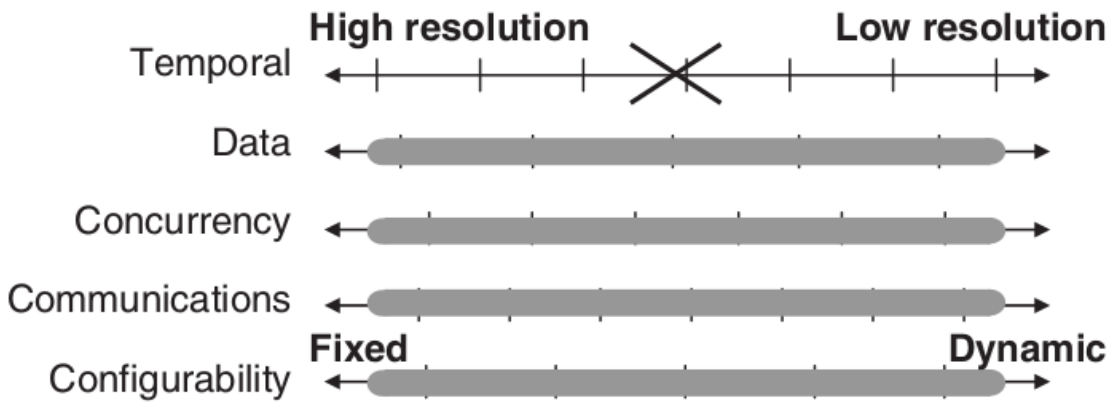
- Dynamically reprogrammable — These devices may be reconfigured with new personalities by loading configuration data dynamically during opera- tion. Dynamically reconfigurable devices can be treated as a resource in the system that can be scheduled and programmed to perform any number of functions. Some types of FPGAs are dynamically reconfigurable.
- This axis is orthogonal to concurrency and communication axis.

ESL Taxonomy Axis



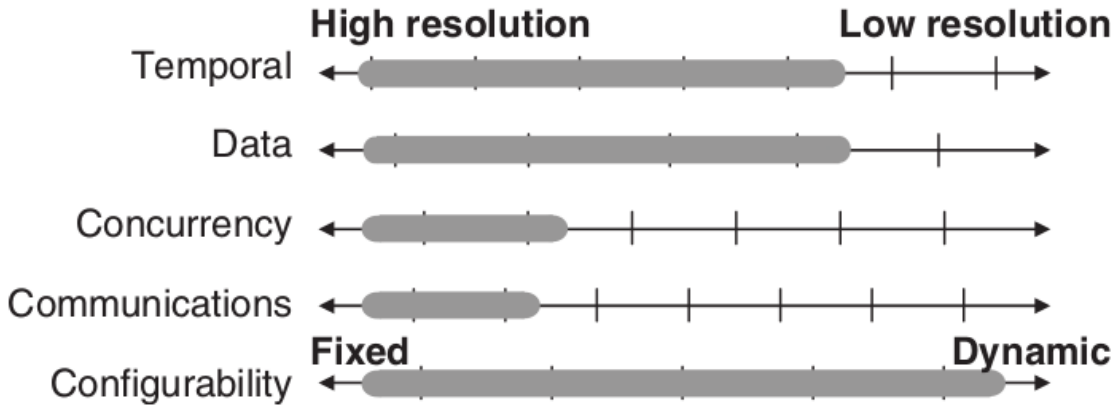
Example - Functional model

- A functional model defines the operations of a system without describing a particular implementation. Functional models can be written at any level of abstraction and can have any of the ESL attributes. The major characteristic of a functional model is that it does not define timing.



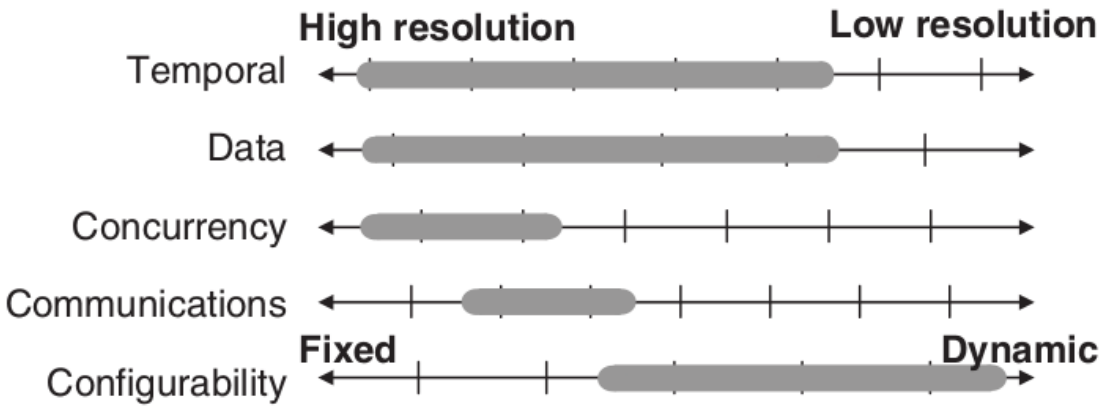
Example - HDL Description

- A pure hardware solution written in Verilog or VHDL is most likely to use point- to-point or buffered communication with tight communication between each of the elements. It is possible to define complete systems with multiple processors using these languages, but in this capacity they are being used as the structural interconnect for building blocks probably defined at higher levels of abstraction. The intent here is the type of input that could be fed into a synthesis tool to produce hardware.



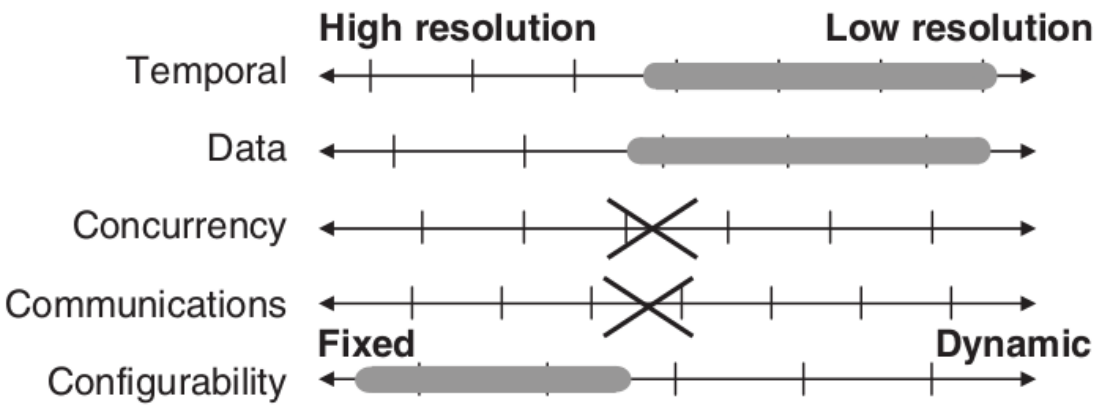
Example - FPGA Solution

- If we were to consider a subset of designs targeted at an FPGA, then configurability is performed either at start-up or dynamically. Inside the FPGA, communication may either be through signals, although this is limited because of timing issues, through buffers, or through local memory. Concurrency is predominantly fine grained.



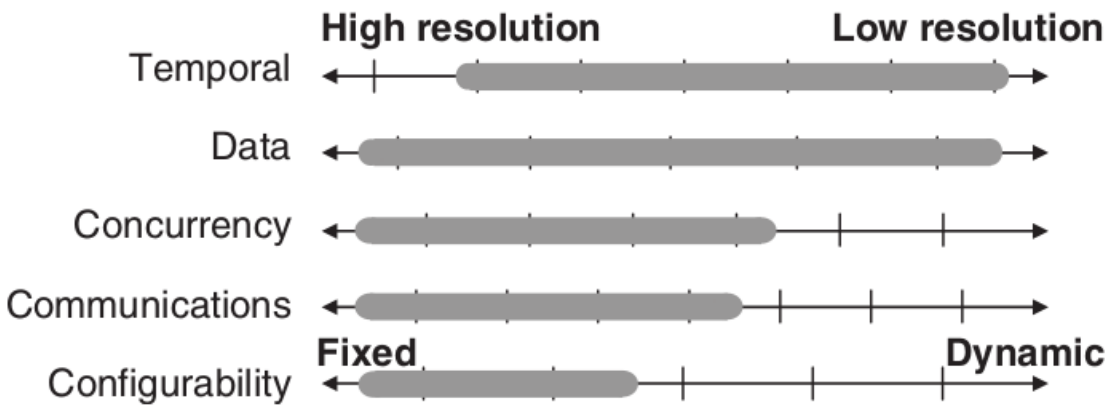
Example - Generic C Solution

- The C language has no built-in constructs for dealing with concurrency or communication. These are capabilities provided by an operating system. Thus, the C language is a highly restricted entry language for concurrent systems. This has led to the creation of many variants of it to deal specifically with concurrency. C has a low resolution when it comes to data rep- resentation, with a variable having a defined value being the closest it can get to hardware. It also has no built-in language constructs for the management of time. Configurability is at the design level because C and C ++ code is not considered to be changeable once it has been compiled and shipped



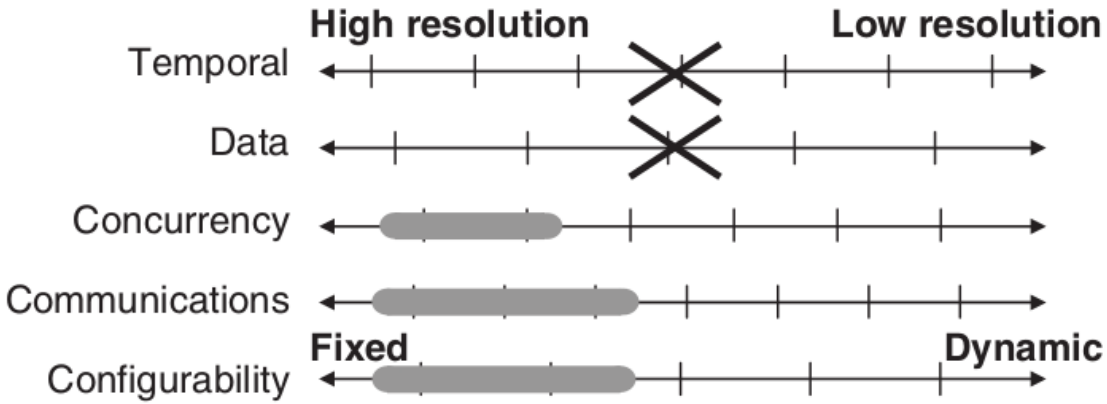
Example - SystemC Solution

- SystemC is popular C ++ derivative. This language adds concurrency and several explicit means of communication, including signal-based and transaction-based as well as messaging through shared variables.



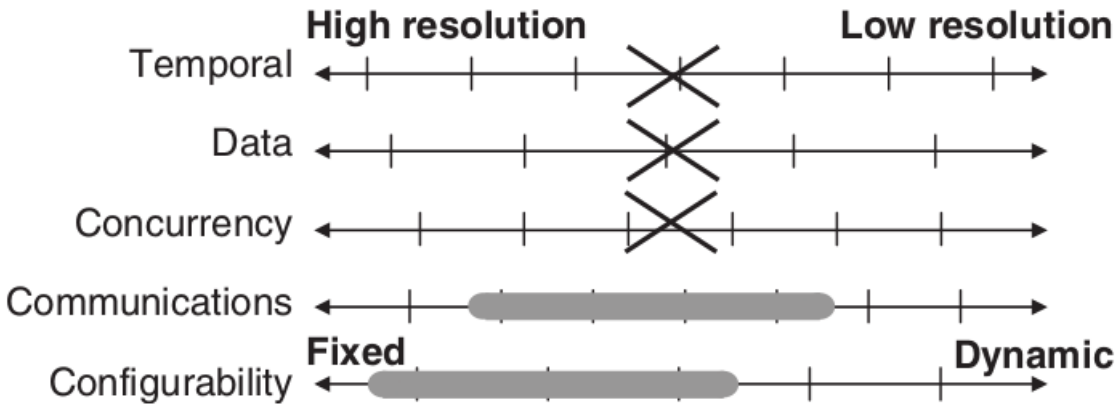
Example - Extensible Processor Solutions

- Many of the tools available today focus on partitioning functionality between a pro- cessor and one or more blocks of hardware that are automatically created. Hardware is used to implement part of the functionality that consumes a large percentage of the execution time. There are a number of ways in which this logic can be attached to the processor.
- In extensible processors, there is a very close coupling where the added hardware becomes an instruction in the processor itself, directly sharing resources with other instructions.



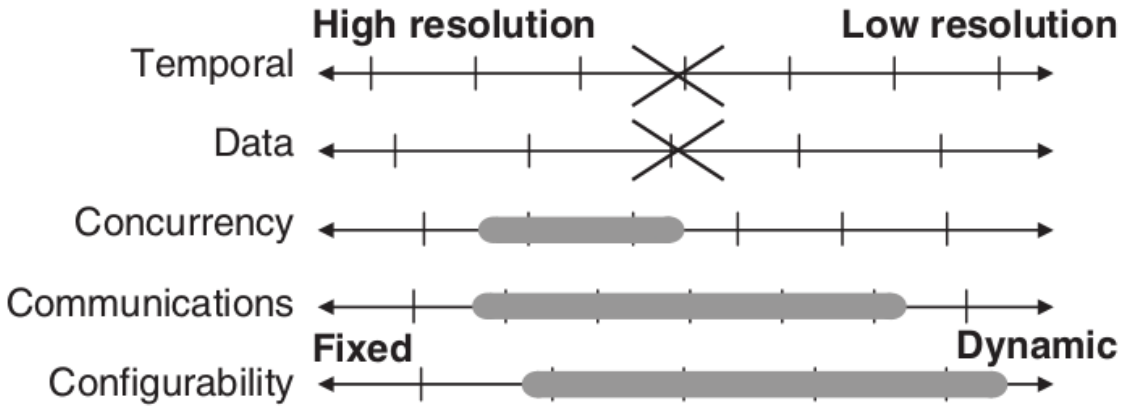
Example - Processor plus fixed-function coprocessor

- Also common are solutions where a coprocessor is created. It is common in coproces- sor solutions for the functionality performed in the processor and the functionality performed in the coprocessor to operate serially, without achieving a great degree of concurrency. Such solutions look at speeding up part of the problem so that it has a significant impact on the total execution time without necessarily increasing con- currency. Alternatively, by reducing total execution time, it may become possible to slow down the main processor so that the gain is translated into lower power.



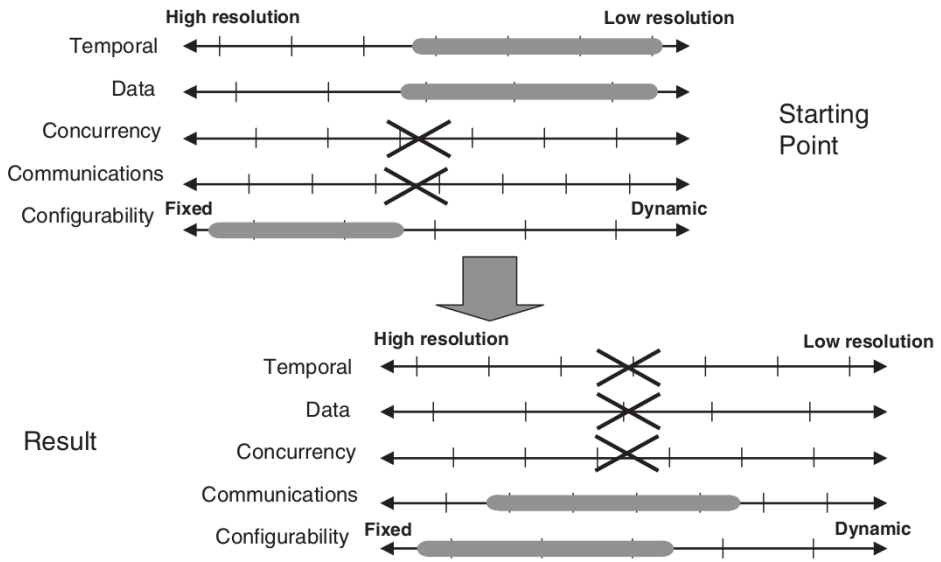
Processor plus programmable coprocessor

- Over time, it is likely that these solutions will improve their capability for taking advantage of concurrency. It is also possible that coprocessors will be generated that are capable of more than one function, giving them greater configurability.



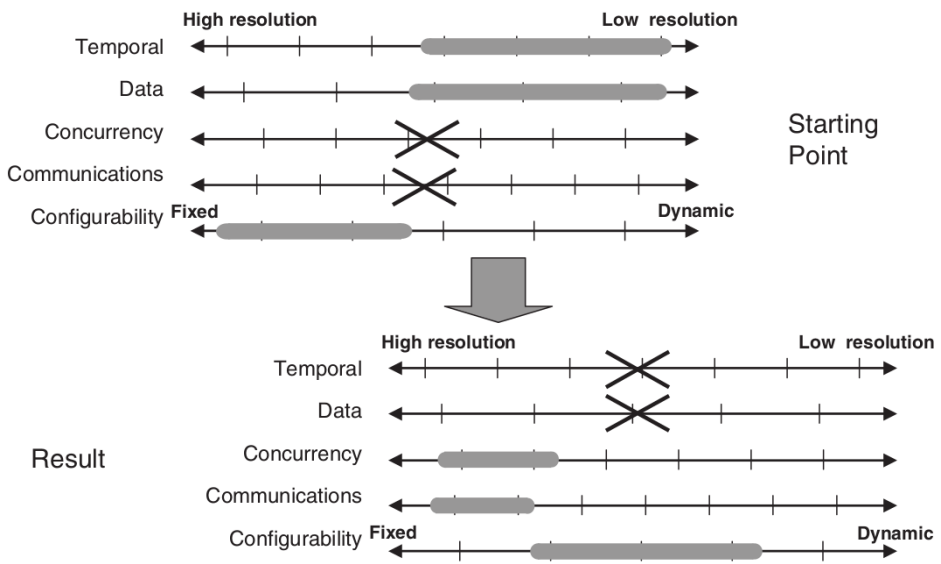
Example Design Flow - 1

- This tool starts from a generic C description and performs analysis on the code, identifying the parts of it that can be migrated from a general-purpose processor onto a highly targeted coprocessor. The solution has the ability to target different levels of communication between the processors, such as memory-based or a direct interface into the processor. The solution shown does not add concurrency—in other words, the main processor is stopped when the coprocessor is running.



Example Design Flow - 2

- The coprocessor created by same tool can have a pipelined architecture, and thus adds concurrency within this processor. It is also possible that multiple functions could be implemented with the coprocessor, with the infrastructure being shared among them.



Example Design Flow - 3

- This tool start from a SystemC description and perform what could be described as more classic high-level synthesis with the aim of producing hardware that can implement the desired functionality.