

Navigating the RTL to System Continuum

Calypto Design Systems, Inc.

www.calypto.com

The rapidly evolving semiconductor industry has always relied on innovation to sustain advancement. From the creative forces behind the myriad of consumer electronic products to the technology improvement behind sub-micron silicon running at gigahertz speed, innovation makes electronic systems possible.

Gartner Dataquest (2004) states that more than half of IC designs are System-on-Chip (SOCs) meaning they contain some type of processor and memory subsystem. With the adoption of IP, from internal or external sources, additional importance is placed on system-level design and integration. Designers are being pushed to work at higher levels of abstraction while at the same time meet strict power and performance requirements. In addition to these pressures, design teams are faced with the familiar challenge of getting their SOC working within a tight project schedule.

It is clear that the semiconductor industry must adopt system-level design and verification methodologies. However, before design teams can move forward there is a prerequisite on tools and technologies that support a RTL to system level transition.

The RTL-System Continuum

System level design is upon us. The adoption of languages like SystemC and SystemVerilog provide access to system-level design techniques. Commercial high-level synthesis tools are emerging to automate the flow from system-level descriptions to RTL. Many commercial simulation tools have recently added transaction-level capabilities that facilitate the use of abstraction. Hardware designers are beginning to work at higher levels. By working at higher levels of abstraction, designers can concentrate on sequential optimizations instead of

laboring through combinatorial implementation. In reality, the move to system-level design is not a “leap” across a large chasm. Rather, it is a series of steps through different “gray levels” of abstraction. During the course of a design, engineers work at multiple levels of abstraction simultaneously. It is here, in the RTL to System level Continuum where design trade offs are made and innovation is brought to fruition.

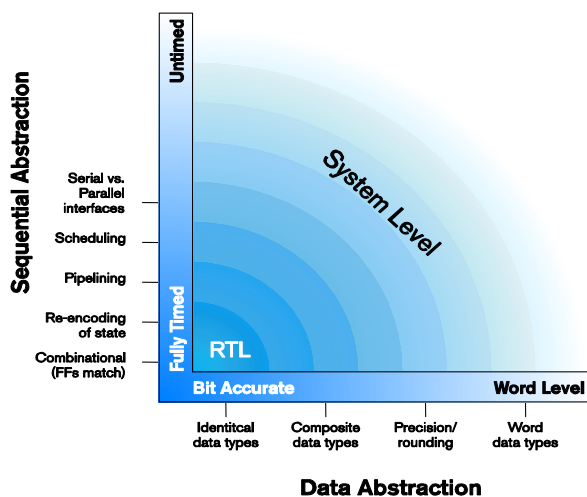


Figure 1: The RTL to System level Continuum

The RTL to System level continuum has two axes of

abstraction: sequential and data. System-level design demands engineers move up in both directions. (Figure 1)

Sequential abstraction ranges from detailed timed description to algorithmic un-timed description. This range can be demonstrated with the example of PCI. As the ubiquitous PC bus for the last decade, PCI has well-defined functionality. PCI64 is a PCI implementation across a parallel bus infrastructure, Where as, PCI Express implements the same functionality with an underlying high-speed serial protocol. At the system level, operations like read and write are functionally equivalent. Yet, PCI Express has a completely different sequential architecture. In this analogy, PCI64 and PCI Express are detailed timed descriptions – the lowest level of sequential abstraction. At a higher level of sequential abstraction PCI transactions have only input/begin and output/end specifying the temporal relation of each command. At the highest level of abstraction PCI functionality is modeled as simple data movement or commands with no notion of time.

Data abstraction shields designers from fine grain detail and allows higher levels of conceptualization. For software engineers the notion of collecting, abstracting and passing aggregated information is part of basic programming. These concepts are just as useful to hardware engineers in addressing complexity. Hardware designers exploiting data abstraction manage “words” and “data structures” instead of “bits” and “busses”. VHDL, SystemC and SystemVerilog all contain type and object semantics to encapsulate data and promote higher levels of data abstraction.

The ability to move within the range of sequential and data abstraction creates the RTL to System level continuum. Higher levels of abstraction allow for design alternatives to quickly be formulated and qualified. The exploration of sequential architectures provides detailed information on system characteristics. Navigating the RTL to System level continuum is iterative. As algorithms are turned into RTL implementation, a series of informed decisions lead to an optimal design.

System-Level Trade offs

For a given algorithm there exist multiple RTL implementations that have a distinct timing, power and area cost. (Figure 2) The most straight-forward implementation: “Design X” does not always meet the system-level requirements. Sequential design modifications must be made. Optimizations that improve design characteristics

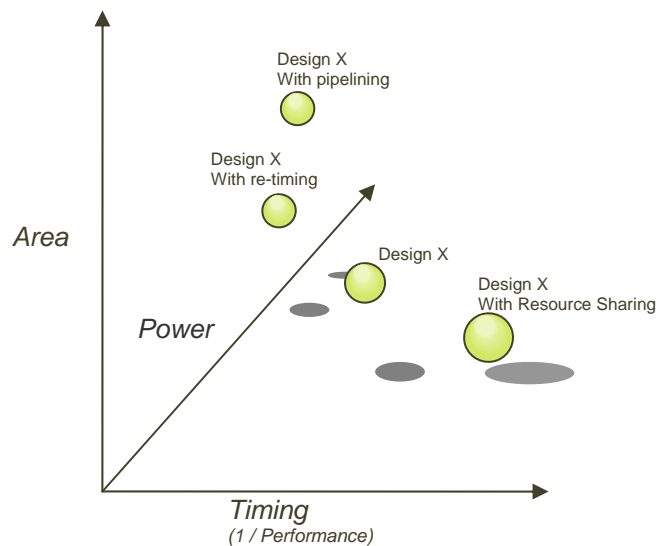


Figure 2: System-level tradeoffs

along one axis come at the expense of other dimensions. For example, improving power in “Design X” by utilizing resource sharing decreases system performance – increases timing. Finding the optimal design implementation is a series of steps through the RTL to System level continuum that incorporates the feedback of previous “what if” scenarios.

A Sequential Shift

Design methodology is changing. This time the change is a sequential shift. Similar to the last significant shift to a HDL based design methodology; the industry requires new solutions to move to higher levels of design successfully. Consider that, even after high-level design languages and RTL synthesis tools were in place, designers continued to combine HDL methods with previous modes of schematic design. Designers gradually gained confidence. During this period, the tools matured and the methodology was refined. And, as evident today, the productivity gains of a HDL based methodology changed the industry.

There is evidence that a sequential shift has begun. Many engineers, regardless of whether they are designing with VHDL, Verilog, SystemVerilog, C, SystemC, or even using behavioral synthesis, transform the sequential behavior of their designs. Why? Because without modifying the sequential implementation there is no way to meet a power, performance, and area budget of their design. Some common techniques to improve operational characteristics are re-timing, resource sharing, and pipelining, all of which modify sequential behavior.

In re-timing, long combinational logic paths between state elements are rebalanced to reduce latency. Microprocessor teams frequently make this change to achieve timing constraints or maximize clock speed. As evident in figure 3A, re-timing transformations are sequential in nature because the values in the re-timed state-map are distorted from the original.

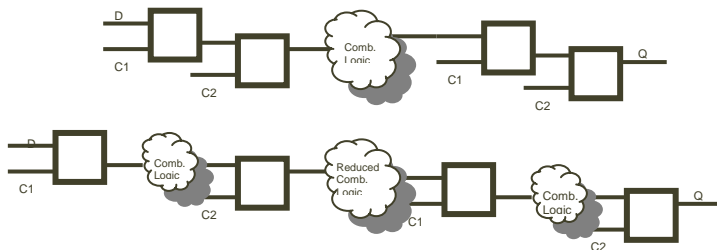
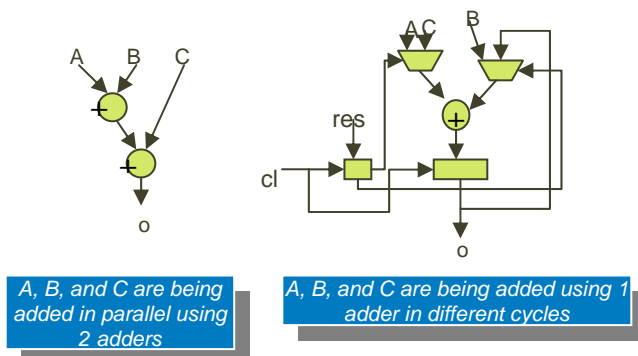


Figure 3A: Retiming

Another example of a sequential change is resource sharing or duplication. In figure 3B, the algorithm $A + B + C$ may be implemented with or without resource sharing. Resource sharing reduces the number of adders in this design at the cost of additional cycles, latency.



A, B, and C are being added in parallel using 2 adders

A, B, and C are being added using 1 adder in different cycles

Likewise logic resources can be duplicated to increase performance at the cost of area.

A technique to improve data throughput is pipelining. Figure 3C shows two data path sequences. In the pipelined, version the “calc” function is broken into two phases so the data path can run in parallel. In this case, the sequential behavior and temporal relationship of the output is altered. Given the designs are functionality equivalent this sequential modification is a tradeoff between latency, throughput and area.

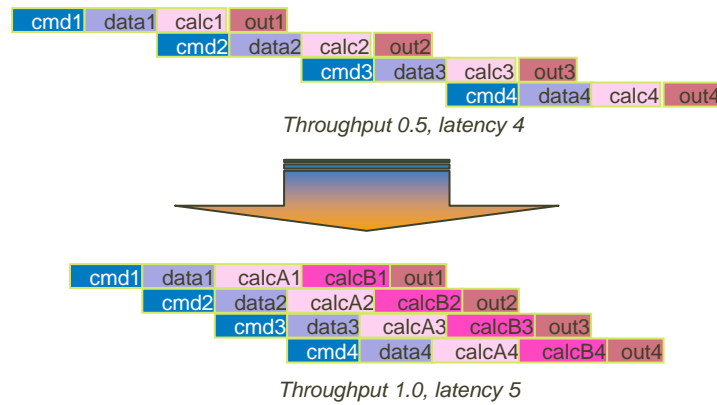


Figure 3C: Pipelining

A sequential shift in design methodology is necessary to effectively design at the system level. System-level design has the greatest affect on performance, power and area in SOC designs. To meet system-level requirements, designers will make micro-architectural optimizations that modify sequential behavior. Today, sequential changes are performed manually. Hardware engineers need a new generation of tools to enable a sequential shift in design methodology.

Verification Limited Design

The semiconductor industry has clear evidence that 70% of development time is spent doing functional verification. Yet, despite its importance, functional errors remain the number one cause for design re-spins.

The cornerstone of functional verification is software simulation. While RTL-based software simulation remains a vital tool, it alone is not sufficient to verify complex SOC designs. It simply takes too much time to create and run the multitude of tests required to fully validate a design. Regressions take several days or weeks to complete. Thus the process of exploring multiple sequential implementations could add weeks or months to a project. Consequently, verification remains the largest barrier to evaluating design alternatives and meeting system level specifications.

To make matters worse, sequential changes can invalidate existing test benches. When this happens, test benches require inspection and adjustment. In the resource sharing example (Fig. 3B), additional cycles in the output are introduced when sharing is implemented. The new output has a clock correlation discrepancy with respect to the other implementation. In this case, the regressions would fail despite equivalent functionality. Depending on the

situation this type of failure could ripple from block level regressions into the sub-system or even system-level.

A verification methodology based on successive refinement from system-level models to RTL-implementation has several benefits. System-level tests run thousands of times faster than RTL simulation. Verifying system-level interactions gives designers' confidence in their architectures and algorithms. As models are refined, previous iterations can be used as a reference. When combined with formal methods, system-level verification efforts are leveraged throughout the design process. This allows quick detection of side effects and insures the implementation remains consistent with the original intent.

Conclusion

The challenges of today's SOC designs have pushed the current tools and engineers to the tipping point. To meet the demand, designers are beginning to work at the system level. They are designing at higher levels of sequential and data abstraction. As design methodology makes a sequential shift a new generation of tools will bridge the RTL to System level continuum. These tools will allow designers to move beyond writing lines of RTL and past verification-limited design.