Silicon Virtual Prototyping

SVP originated in the ASIC world to deal with *mega-designs* (designs with 10s of millions of logic gates).

The problem with *traditional flows* is that many design issues **do not** become apparent until accurate timing analysis is performed.

This cannot be done until the entire flow is finished, from synthesis to place-and-route.

Any problems discovered at this point force an iteration, that is time consuming and can delay the time-to-market of the product.

One way to allow problems to be identified quickly is to create an ***SVP***.

An *SVP* is a representation of the design that can be *generated quickly* but contains sufficient information for designers to identify problems.

In theory, design iteration time using SVP can be measured in *hours*, as opposed to days or weeks using conventional design flows.

ASIC-based SVP Approaches

The role of *logic synthesis* is to accept an RTL representation of the design along with a set of timing constraints.

RTL is converted into a mixture of registers and Boolean equations (after performing optimizations and minimizations) and then a gate-level netlist.

Hopefully, the netlist meets the *original timing constraints.*

Conventional logic synthesis tools operate in the *gate-size* vs. *delay* plane.

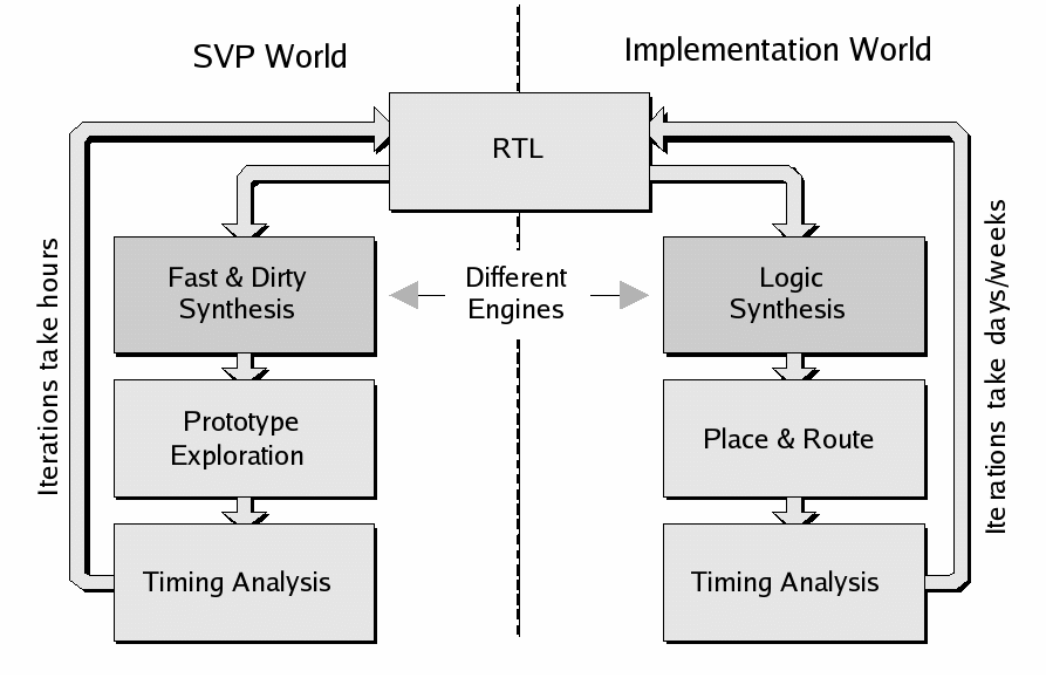
These tools expend tremendous amounts of time and effort to evaluate trade-offs between gate size and corresponding delays.

Unfortunately, many of the decisions are often rendered meaningless during the place-and-route portion of the fl ow

Gate-level SVPs perform a *rough-and-ready* placement of the gates using a *fast-and-dirty* synthesis engine.

ASIC-based SVP Approaches

The *fast-and-dirty* engine is usually based on completely different algorithms than those used by the full-blown synthesis engine.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

The gate-level netlist of the *SVP* is not as accurate as the final version.

Once engineers have performed *RTL exploration and timing analysis*, a full-blown, *physically aware*, logic synthesis is performed to generate the *real* netlist for the P&R tools.

ASIC-based SVP Approaches

The fact that different tools are used in each of these fl ows can lead to poor correlation of the results, and may require *back-end-to-front-end iterations*.

In other words, the whole purpose of SVP is defeated.

In '99, Sutherland, et al. proposed a **gain-based** synthesis of gate-level *SVP*s.

Here, *logical effort* concepts are used to establish a **fixed timing plane**, that the subsequent P&R tools work within.

This means that ALL timing optimizations are completed and all circuit delays are determined and **frozen** by the end of synthesis.

For placement, the engine uses a *size-driven* algorithm, where all gates are *dynamically sized* to meet their timing budgets, based on their loads.

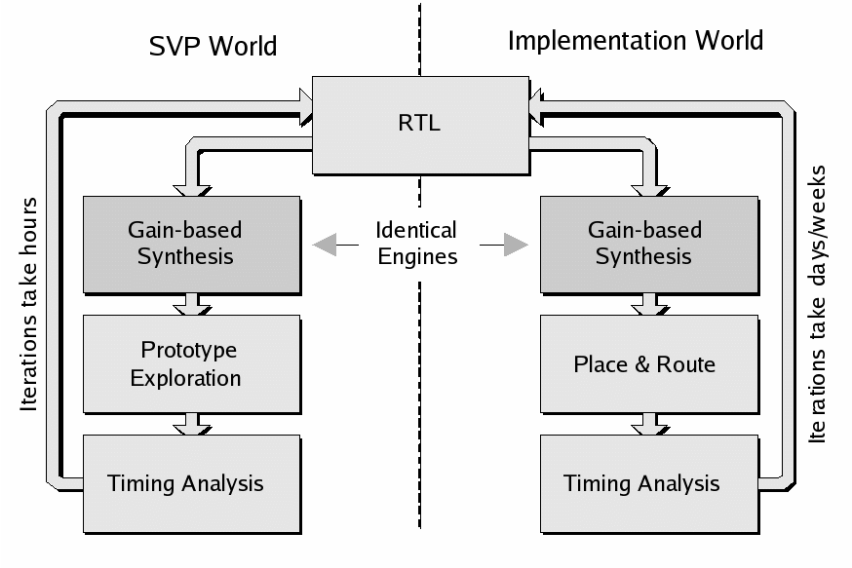
For routing, the load-driven engine *tunes* the width and spacing of the tracks to maintain the timing budgets and to ensure signal integrity.

ASIC-based SVP Approaches

This *gain-based* approach automatically uses up any **slack** in the path delays by using the smallest sized gates possible.

This yields a smaller footprint and significantly reduces congestion, power consumption and noise problems.

Most importantly, the increased speed and capacity of the *gain-based* approach allows the *same* synthesis engine to be used for both prototyping and implemention.



This obviously provides high correlation of the end results and reduces or eliminates back-end-to-front-end iterations.

The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Cluster-based SVP Approaches

Most of today's *SVP*s are based on full-blown *gate-level* netlist representations.

Even though these representations are generated using *fast-and-dirty* synthesis, the millions of logic gates they posses may strain *SVP*s placement and analysis engines.

Clustering can be used to help *SVP's* placement decisions and track-delay estimations.

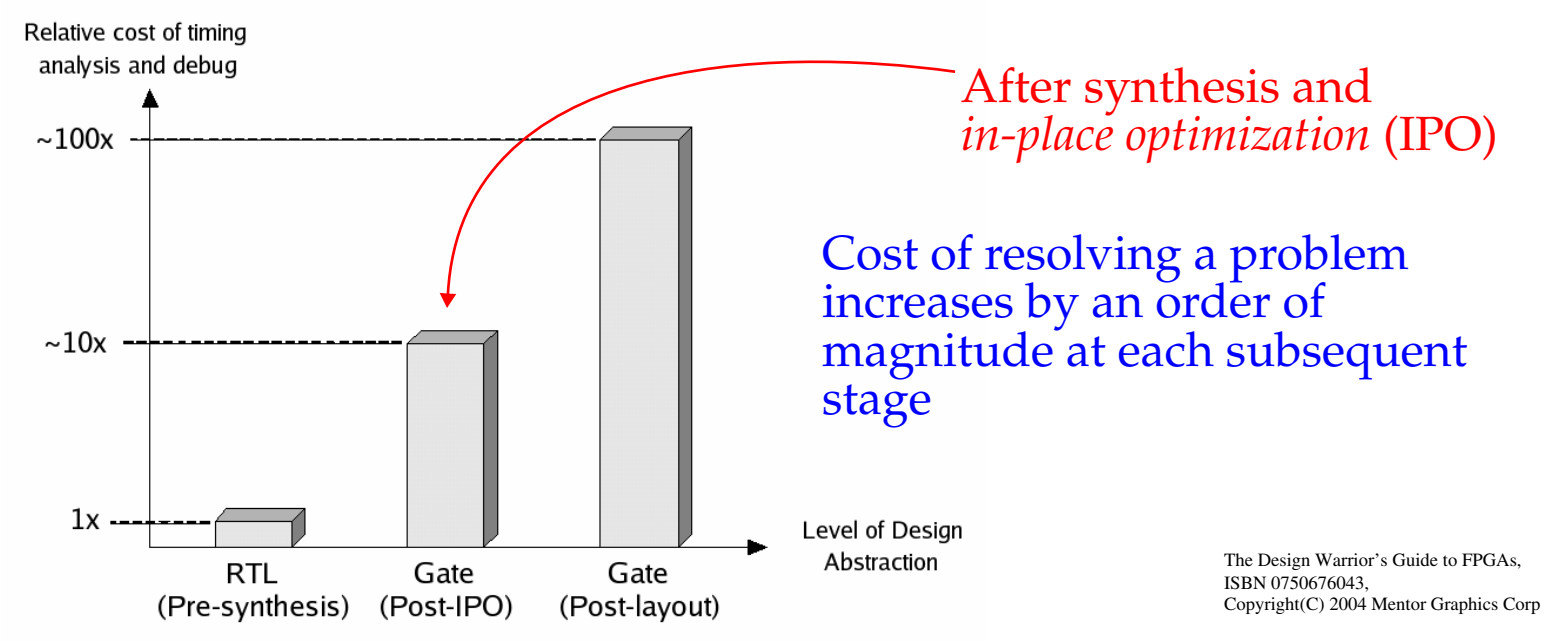
Here, the registers and gates produced by *fast-and-dirty* or *gain-based* synthesis are gathered into groups or *clusters*.

Cluster size ranges from 10s to 100s of cells, which allow them to remain small enough to preserve placement quality.

More importantly, the reduction in the number provides a significant improvement in the run-time of the tools.

RTL-based SVP Approaches

There are *three* major breakpoints in the design fl ow of digital ICs wr.t. analyzing timing, area, etc.



Timing closure refers to analyzing a design or architecture to detect and correct any problematic timing paths.

It is an iterative process that may require several iterations before convergence is achieved.

RTL-based SVP Approaches

Obviously, *post-layout* timing analysis is the most accurate, but also the most painful, so designers avoid making changes at this level.

Relatively accurate timing analysis is possible at the gate level following synthesis and IPO (middle bar on previous graph).

Unfortunately, getting here using conventional fl owsrequires the use of *physically aware synthesis* to provide a *placed* gate-level netlist.

As we discussed, these are compute-intensive and time-consuming.

Gate-level *SVP* is of some help, as we discussed, but the process can still be compute-intensive and time-consuming.

A better approach is *RTL-based SVP*, to allow designers to quickly identify and address paths that will cause downstream timing problems.

RTL-based SVP begins by taking the *logical* and *physical* (LEF and DEF) definition files of an ASIC cell library and generates a **design kit database**.

RTL-based SVP Approaches

The *design kit* is a database of characterized *logic functions*, such as counters, XOR trees, etc., (not gates) that is used directly by the *RTL-based SVP*.

The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

The *design kit* captures the behavior of these logical functions, including timing and area estimations.

The *RTL-based SVP* accepts the RTL code of the design, the timing constraints in SDF and the *design kit* of the target cell library.

The *SVP generator* converts the RTL into a netlist of *work functions*, which are abstractions of elements in the *design kit*.

Once converted, the *SVP generator* performs gate-level optimizations, common subexpression elimination, constant propagation, loop unraveling, etc.

RTL-based SVP Approaches

The *SVP generator* then performs a *virtual placement* of these functions, which is used to generate accurate area estimates, followed by timing estimates.

Proponents of *RTL-based SVP*s claim a 40-fold speed increase over post-IPO generation and analysis engines.

Also, supporters claim that timing analysis results correlate with post-IPO delays within an error of 20% or less (which is not bad).

FPGA-based SVPs

The same place, route and timing analysis problems are occurring with multimillion gate FPGA designs.

One problem in particular deals with the fact that FPGA P&R tools work with a *flattened* representation of the design.

Making a small change in a single block of the RTL code results in a re-run of P&R on the **entire design**.

FPGA-based SVPs

Some FPGA vendors are now supporting the concept of an FPGA *SVP* and the ability to perform P&R on a individual design blocks.

FPGA SVP involves using a graphical, top-down tool that shows all of the logical resources, e.g., LUTs, CLB, multipliers, of the target FPGA.

After logic synthesis, the *SVP* generator loads the hierarchical LUT/CLB-level netlist and automatically generates a fl oorplan.

The fl oorplan contains square and rectangular blocks that correspond to top-level modules in the design.

The editor allows ***interactive manipulation*** (resizing and movement) of the squares and rectangles to deal with any timing problems.

Once completed, you can select one or more blocks and run P&R.

The tool preserves portions of the design that are P&R'ed.