# System design

*Andrew Brown, Bashir Al-Hashimi, Mark Zwolinski and Peter Wilson*
*Department of Electronics and Computer Science*
*University of Southampton*

## Preamble

Designing big systems is difficult, making them work is arguably an order of magnitude more so. Why is this? The problems are not fundamental; rather the difficulties are the design/implementation equivalent of a death of a thousand cuts. Sometimes this is called the "Design Gap", and usually shortly thereafter the words "timing closure" appear in some guise or other. Is this a "Grand Challenge"? Our view is that the real problems are largely political and economic, rather than technical or fundamental, and that this is one of the reasons that the situation is not yielding to piecemeal technical fixes to individual problems. (We're not saying there *aren't* difficult technical problems. It's just a question of scale.)

*Is this discussion/proposal a "common vision"?*
It lies outside the scope of Steve Furbers' document because it is about big software and device modelling is an important and intrinsic part of this. But then it's hard to imagine a useful EDA tool that isn't big and doesn't rely on some sort of model somewhere along the way, so we think the scope is flawed here.

*Is it just a large responsive mode proposal?*
It is difficult to see how it could be cast into a form acceptable to EPSRC. It is hardly research, and it's not even development. It's more re-development, and the only argument we can offer against the multitudinous cry of "Industry should be doing this" is the observation that they aren't.

## The design flow

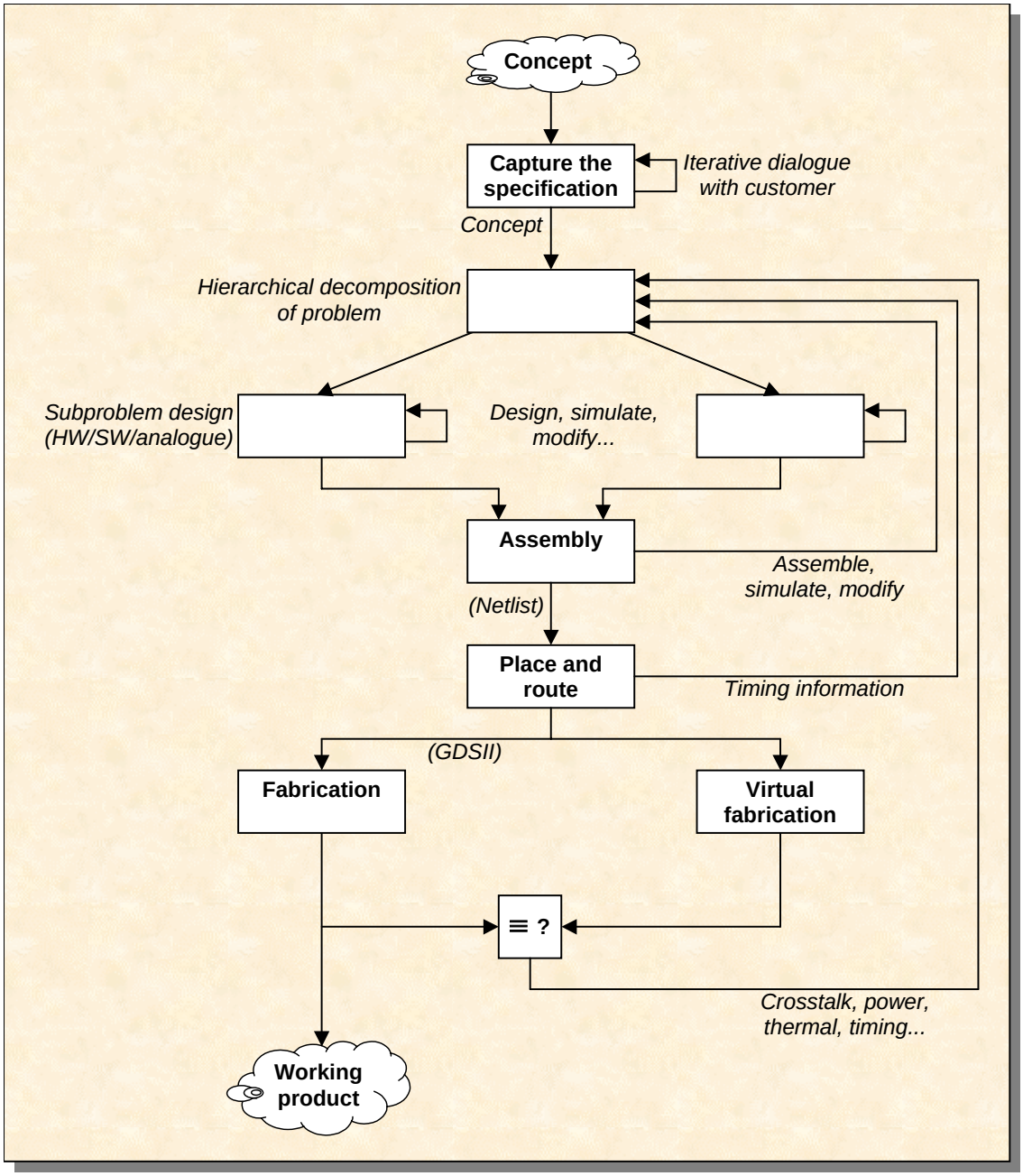
More or less - is shown in the figure. It is idealised.

*Capturing the specification*
Usually comes down to human conversations between the customer and the design house. Numerous attempts to automate the process have been attempted; none have received acceptance.

*Hierarchical decomposition of the design problem*
This facilitates the mapping of design resources (usually people) to (manageable) tasks, and supports parallel development.

*Assembly*
Well, someone has to. The 'output' at this stage is a netlist - a topological description of the electrical connectivity of the system - which may well be - probably is - hierarchical.

*Automated place and route (APR)*
Leaving aside programmable platforms (FPGAs), the netlist is turned into *geometry*: things (devices, blocks, wires) have shape, size and position. (And in acquiring these attributes, a whole lot of undesirable parasitics - crosstalk, inductance, capacitance and resistance are also introduced, which it may or may not be safe to ignore.)




*Fabrication*
Finally, the circuit is fabricated, where lots of very expensive machines guarded by very strange people blow nasty chemicals over bits of glowing semiconductor, which produces an *approximation* to what you think you're getting.

*And finally*
Out pops a chip that does what you want.

## Why is it like this?

The snappy answer is that it works (or used to) and no-one can think of a better way. But there is also an element of evolution and inertia: thirty years ago circuit designers used to cut out shapes in Rubilith and in some cases push the silicon into the furnaces themselves. Circuits were small and devices were big, such that a transistor could be sensibly analysed as a dependent current source, and wires were instantaneous,

bidirectional and lossless interconnect. Devices got smaller, specifications (and therefore systems) got bigger, and humans get overwhelmed by scale (which is *not* complexity). The overall design problem - quite rightly - is chopped into smaller problems, and the smaller problems attacked by design automation algorithms.

### What's wrong with that?

- The interaction between the small problems is no longer negligible; it dominates the behaviour of the blocks themselves. Almost nothing can be done in isolation any more; the big system needs feedback loops.

- The computational cost of the feedback loops. The little ones happen. They have to, and they're cheap. The bigger ones simply don't, because industry can't afford it - or in many cases now it's not a question of money, it is simply that the computational resources necessary do not exist (and will never do so). Ariane 5 and the Pentium V bug "escaped" ultimately because the developers couldn't afford to simulate everything, and they guessed wrong.

- The lack of automation of the feedback loops. The "big picture" is still usually manually controlled, because the tools necessary are not integrated.

⇒ Fabrication level artefacts and parasitics must be modelled at the system level and fed back up to the highest design level.

⇒ Automating this outermost loop will enable the feedback to kill the timing closure, crosstalk, power dissipation ... problems.

A hypothesis (which fits observed data): EDA companies exist to make money, and they put resources (human expertise) where it will do most good. To automate the figure, you need

- Expertise/experience in *all* the areas of the figure. (Most EDA companies don't have this.) A big portion of an EDA companies efforts goes into sales, maintenance and interoperability, and if a technology is mature (as APR, for example, is) you don't keep the development teams intact, hanging about. You get hem onto something else, or maybe they leave, or retire....

- A clear market pull to justify the investment. The user community needs to be able to sit still long enough to think about it, then raise a unified clamour to tell the EDA vendors what it wants. It's easy to see why this doesn't happen. (The phrases "fire-fighting" and "herding cats" spring to mind here.) Equally, or to put it another way, industry is absorbed in solving todays' problem before their competitors do, not trying to get EDA vendors to make a massive investment that will deliver nothing to anyone for years.

- and this is why it isn't happening.

### Get to the point:

We would like to build a behavioural synthesis system that encapsulates and automates the *entire process* of the figure. It should

- Allow the user to describe the functionality of the design in terms natural and sympathetic to her/him.

- Allow the user to assert - with varying degrees of importance - non-behavioural abstract system parameters such as overall power dissipation, area, side channel leakage, testability and so on.

- Automatically explore the design space defined by the behaviour and the design parameter assertions. The space will include design decisions with a wide spectrum of granularity.

⇒ Parts of the system may be implemented as GALS, asynchronous, software in embedded core, pre-defined high-level library cells, specifically synthesized processor cores, whatever.

⇒ *Individual wires* may be re-routed or buffered to overcome crosstalk effects and signal bounce.

And the key point is that the user neither knows nor cares how the design is created, any more than a programmer needs to worry about the interrupt stack or RISC/CISC when they create the ubiquitous "Hello world" program.

### Back to the guidelines

*Significance:* Yes to everything expect the bit about the general public.
*Scale:* Yes to everything, but depending on how you define 'multidisciplinary'.
*Timeliness:*

- Are the objectives... Yes.

- Why is it feasible... Computers with sufficient power to handle the massive design databases required are becoming accessible even to University researchers. This was not the case even five years ago.

- What are the risks...The heart of all this is the single fundamental tension between model accuracy and the computational cost of analysis. This has to be coerced to fit inside the limits of what computers can ultimately do within a realistic (useful) timescale. But success is not a binary decision: with the infrastructure outlined in place, individual point attacks on the bottlenecks can keep us all quiet for years...