EDA - Electronic Design Automation or Electronic Design Assistance?

NSF Workshop Electronic Design Automation
Past, Present, and Future

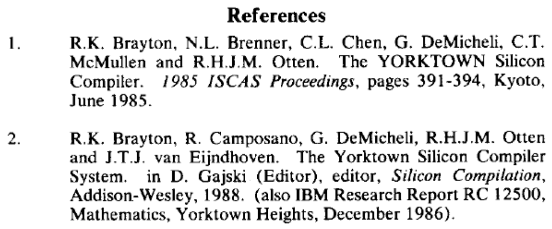
Andreas Kuehlmann

cādence™
RESEARCH
LABORATORIES

NSF Workshop, July 8 – 9 2009

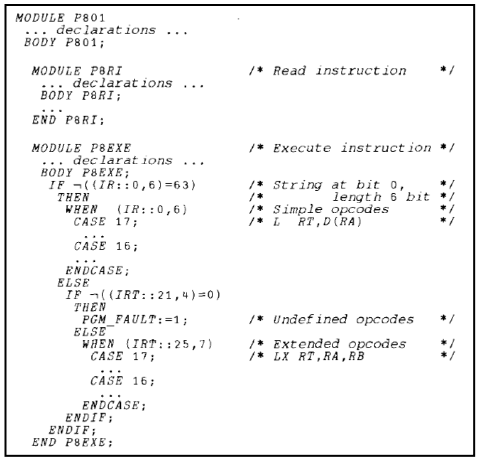
The Past

- The Vision of the Silicon Compiler
  - Similar to SW compilation, simply describe a spec of your hardware and leave the implementation to a tool
- Assumption:
  - It is “just” an optimization problem and with enough heuristics and tricks we will get close to an optimal solution
- Reality:
  - Today we have hundreds of EDA tools stitched together with millions of awk, perl, sed, tcl, and sh scripts



References

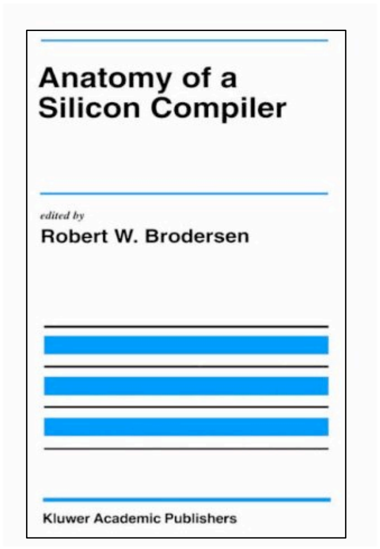
1. R.K. Brayton, N.L. Brenner, C.L. Chen, G. DeMicheli, C.T. McMullen and R.H.J.M. Otten. The YORKTOWN Silicon Compiler. *1985 ISCAS Proceedings*, pages 391-394, Kyoto, June 1985.
2. R.K. Brayton, R. Camposano, G. DeMicheli, R.H.J.M. Otten and J.T.J. van Eijndhoven. The Yorktown Silicon Compiler System. in D. Gajski (Editor), editor, *Silicon Compilation*, Addison-Wesley, 1988. (also IBM Research Report RC 12500, Mathematics, Yorktown Heights, December 1986).



```
MODULE P801
 ... declarations ...
 BODY P801;

  MODULE P8RI                    /* Read instruction    */
   ... declarations ...
   BODY P8RI;
   ...
  END P8RI;

  MODULE P8EXE                   /* Execute instruction */
   ... declarations ...
   BODY P8EXE;
    IF ¬((IR::0,6)=63)           /* String at bit 0,    */
     THEN                        /*       length 6 bit  */
      WHEN  (IR::0,6)            /* Simple opcodes      */
       CASE 17;                  /* L  RT,D(RA)         */
        ...
       CASE 16;
       ...
      ENDCASE;
     ELSE
      IF ¬((IRT::21,4)=0)
       THEN
        PGM_FAULT:=1;            /* Undefined opcodes   */
       ELSE
        WHEN (IRT::25,7)         /* Extended opcodes    */
         CASE 17;                /* LX RT,RA,RB         */
          ...
         CASE 16;
          ...
        ENDCASE;
      ENDIF;
    ENDIF;
  END P8EXE;
```



Towards the Present

- In two-level synthesis life was easy:
  - Objective function:

$$w_a \cdot area + w_d \cdot delay + w_p \cdot power \rightarrow \min$$

  - Everything coincided pretty much with minimizing cubes and literals

$$\#\, cubes \rightarrow \min$$

$$\#\, literals \rightarrow \min$$

- In full chip synthesis from RTL to GDSII this has changed:
  - Objective is still similar:

$$w_a \cdot area + w_d \cdot delay + w_p \cdot power \rightarrow \min$$

  - But we have thousand and thousands of side constraints coming from a maze of heuristics at each stage:

$$f(area, delay, power, a_1, \ldots, a_{10000}) = 0$$

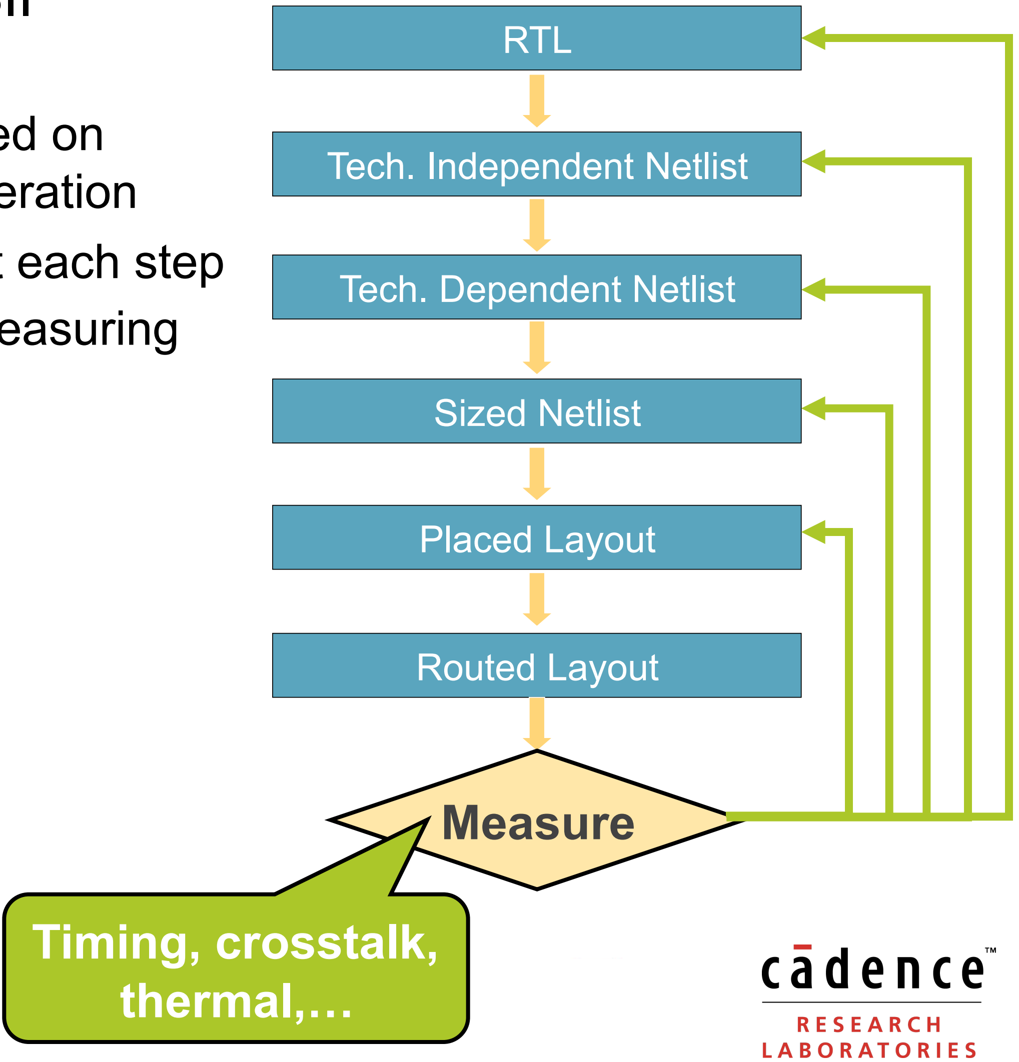
  - The structure of *f* is not fully known and the a_i are all hidden variables,

cādence™ RESEARCH LABORATORIES

Today's Design Flows are Split into many Steps

- There is **no** monolithic optimization approach to the RTL-to-GDSII synthesis problem
  - Traditional approaches based on horizontal flow slicing and iteration
  - Simplified models applied at each step
  - Optimization achieved by measuring and readjusting weights
- There is **no** guarantee of convergence!
- There is **no** guarantee of incrementality

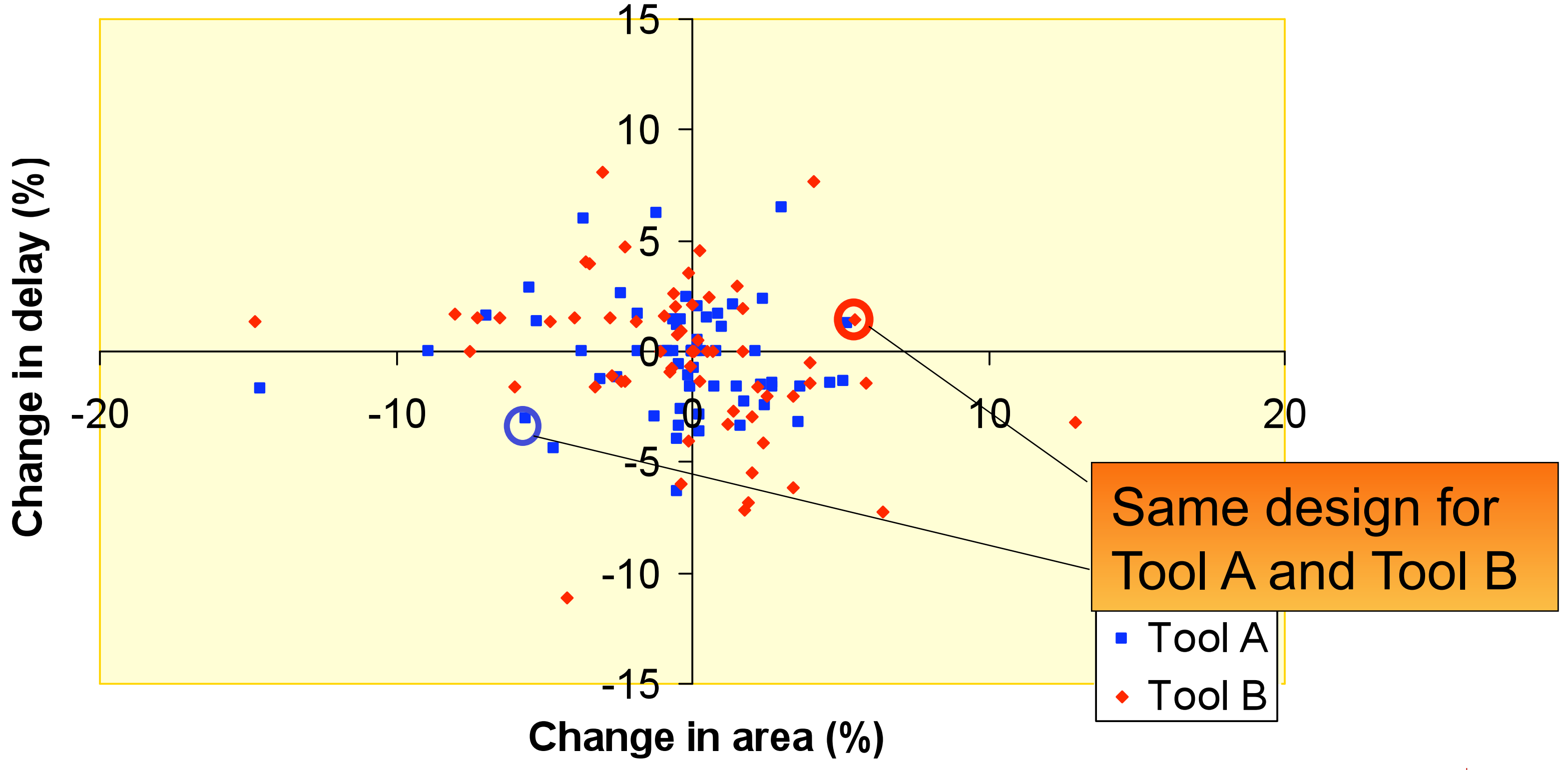
- Result:
  - Extremely instable flows



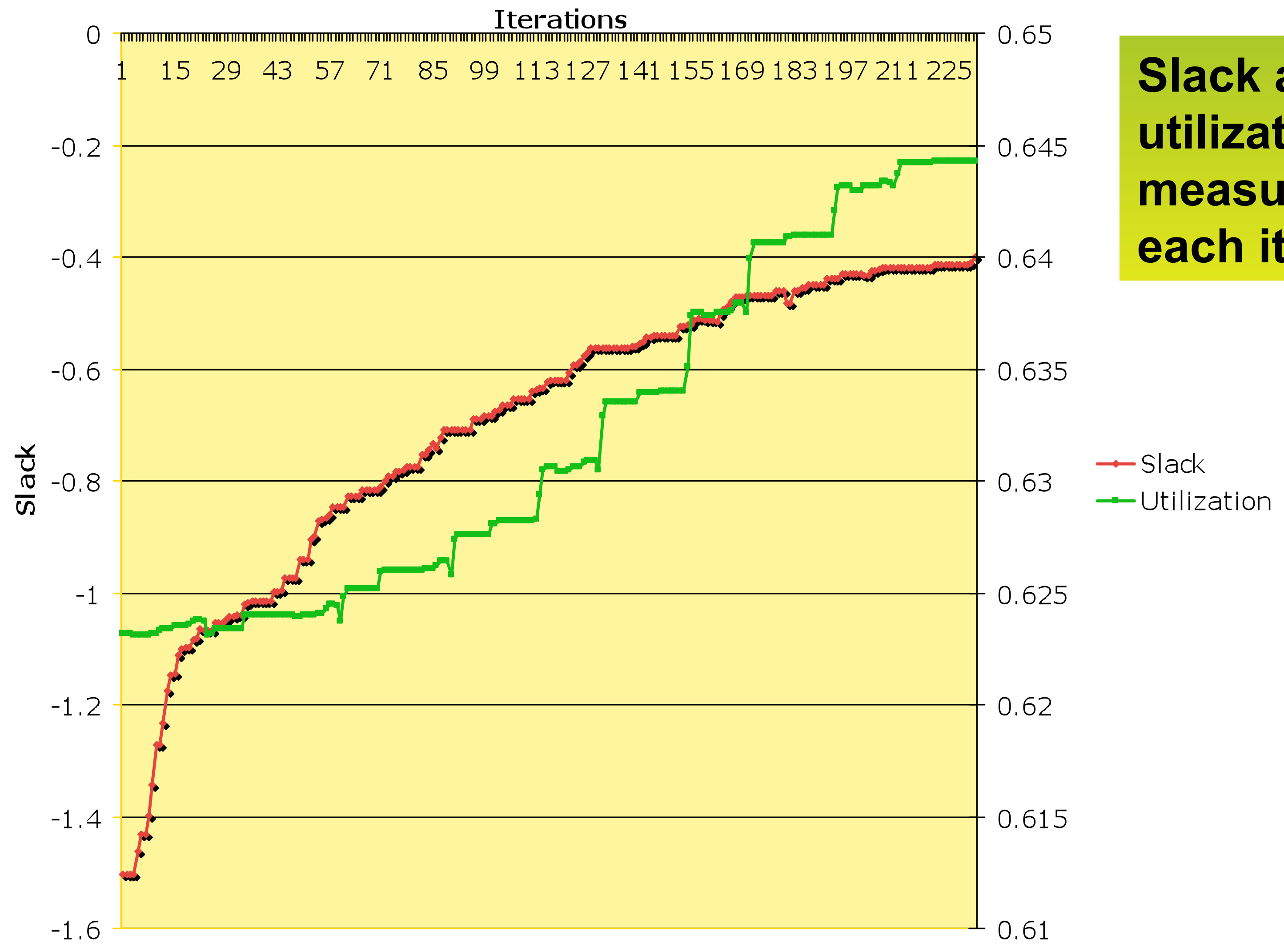
cādence
RESEARCH LABORATORIES

Example: Instability of Logic Synthesis

- Logic synthesis experiment with public benchmarks (IWLS 2005)
  - Original RTL synthesized versus identical RTL with names mangled



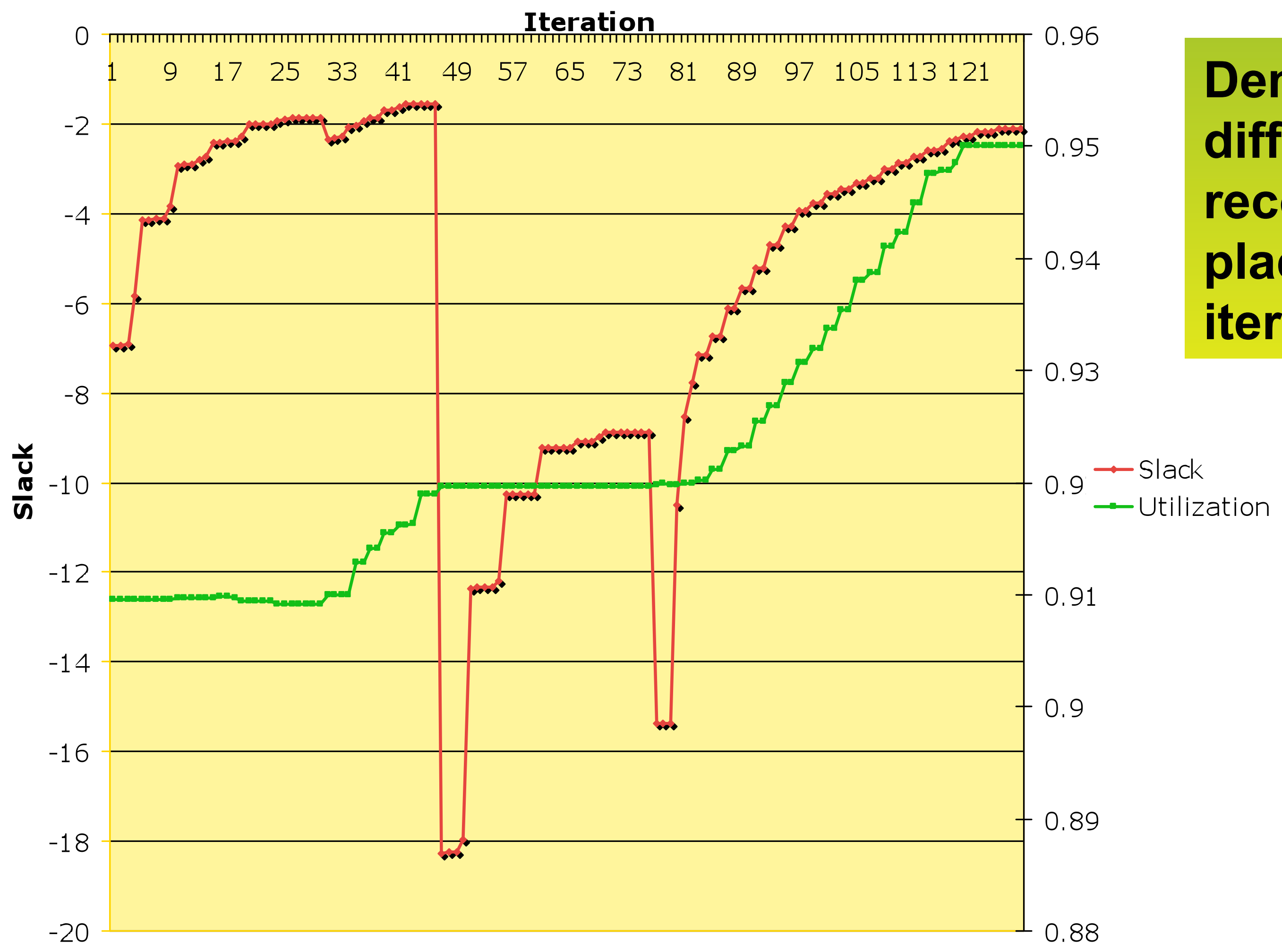
Example: Timing-driven Placement with Sizing and Buffering



Slack and utilization measured after each iteration

kuehl@cadence.com

Example: When Things Don’t Go as well as we Thought



Demonstrates difficulty to recover from bad placement iteration

The Present

- Tools are highly complex with hundreds and hundreds of options to control the flow and influence the results
  - Mutual influence often unknown, non-intuitive, and contradicting
  - Designer mostly revert to “incremental adjustments” of previous flows
    - “Don’t touch it if it somehow works”
  - Optimize design by playing with options and source code structure
    - It is like a Monte Carlo simulation with only 3 samples
  - The raise of the “Application Engineer”
- Tool development very complex
  - Needs to be “backwards compatible” with previous results
  - Peephole improvements often no affect or noisy – hard to innovate on single algorithms
  - Solution: “Add another option to the tool”
- Increasing disconnect between “crisp formulation of research problem” and impact in real flow
  - Makes transfer of University research increasingly difficult

Vision

Reality

cādence™ RESEARCH LABORATORIES

The Future

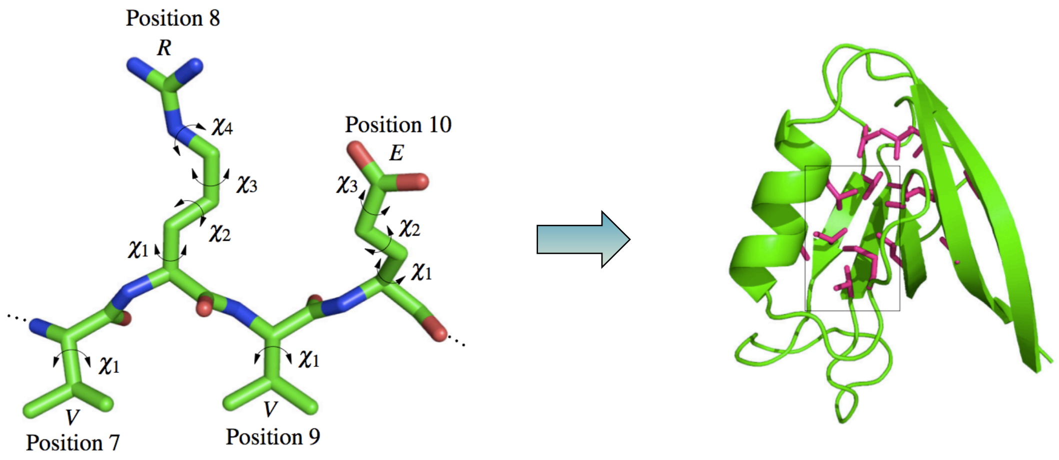
- Core EDA (RTL to GDSII)
  - Remove as many a_i's as we can
    - Can dramatically improve results
  - Truly deterministic design flows
    - Statistical design by running 10.000 placements?
  - Support of highly derivative design flows
    - Economics will continue to reduce the design starts
    - Delta-synthesis, delta-verification, ….
  - Software development technology
    - Legacy software
      - How do we innovate large SW projects with millions of lines of code that need to be backward compatible and have a huge parameter space?
    - The end of the era of the RAM
      - There is no such thing as random accessibility anymore
    - Use of parallel platforms

The Future

- System
  - How do we raise the level of abstraction and still have a route to an efficient implementation?
    - Quick synthesis to layout into inner loop?
      - Requires a deterministic flow!

- DFM
  - How do we shield the design flow from the manufacturing constraints?
    - Is there a new signoff interface above GDSII?

- … or using our EDA experience in other fields
  - DNA sequencing, protein folding,



Thank you!