

*Chip Genesis*



# Creating Chip Generators for Efficient Computing at Low NRE Design Costs

Ofer Shacham (Stanford University / Chip Genesis Inc.)  
Professor Mark Horowitz's VLSI Group At Stanford

*High Performance Embedded Computing (HPEC) 2011*

September 22, 2011



# My Goal Today Is...

- Explain what a '*Chip Generator*' design methodology is
  - Why it is needed
  - Why it solves many problems
- Get you all to download and experiment with *Genesis2* – a prototype tool for creating chip generators
  - It's a "*generator for generators*"
  - <http://genesis2.stanford.edu/>

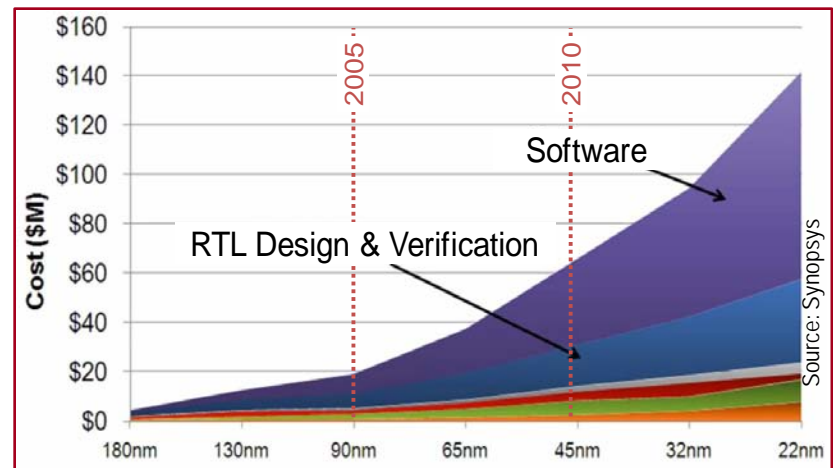


# Disruptive Forces Changing The IC Industry

- Power is today's greatest design constraint
  - Power no longer scales with technology
  - Hand-held/embedded devices drive the market
  - At a fixed power budget: more ops/sec requires less energy/op.

The need for customization has never been greater

- Complexity is growing
  - Number of devices/chip still scaling
  - Algorithmic complexity grows
  - NRE costs are out of control



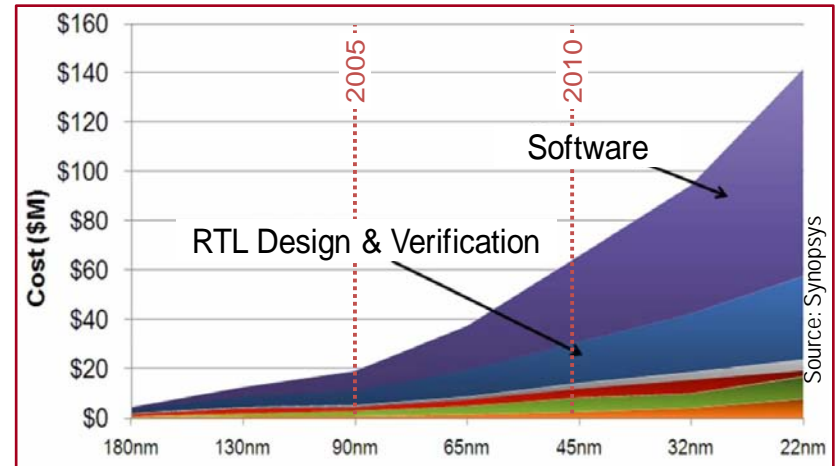
Fewer and fewer applications have markets big enough to justify the effort



# The Biggest Issue

- The biggest issue is NOT design
  - And also not masks
- The biggest issues are
  - System firmware (i.e., software)
  - System validation
  - System optimization

**How the system works**



- Our systems are so complex that they are very hard to reason about
  - Building complex systems is... complicated. And we need to face that fact!
  - We need not to encode the system, we need to encode the reasoning

We'll get back to this a little later

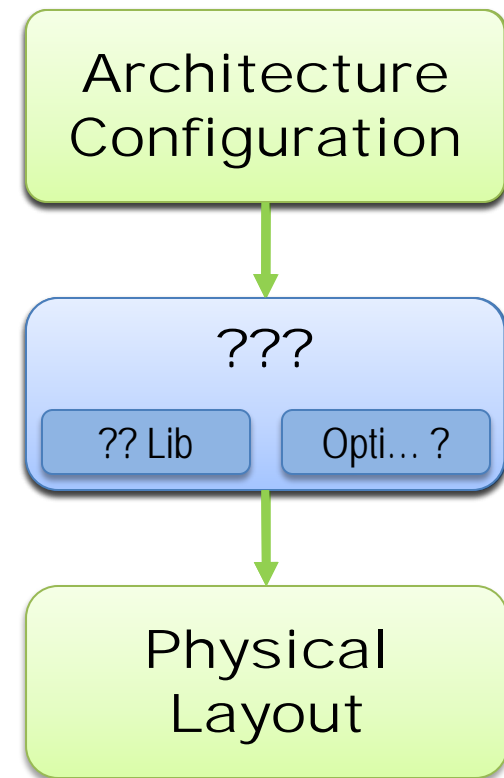


# A Wise Man Once Said...

## Platform-based Design – Alberto Sangiovanni Vincentelli, 2001

Integrated circuits ... will most likely be developed as an instance of a particular (micro-) **architecture platform**. ...they will be derived from a *specific "family" of micro-architectures*, possibly oriented toward a particular class of problems, that can be modified... by the system developer. (pg 6)

An **architecture platform instance** is derived from an architecture platform by **choosing a set of components from the ... library** and/or by **setting parameters of re-configurable components** of the library. (pg 7)

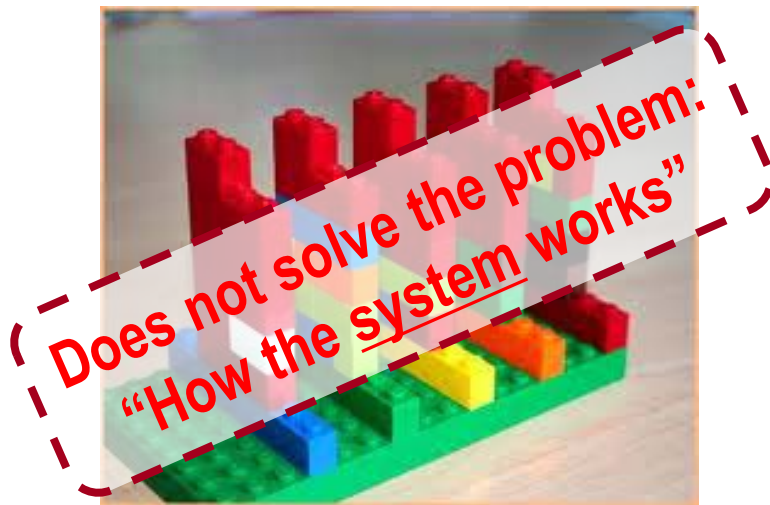




# Industry Went Down Two Roads

## System-on-Chip

- 'Random' configuration of pre-defined, pre-verified, and fixed in Verilog cement, IP blocks



## Re-configurable System

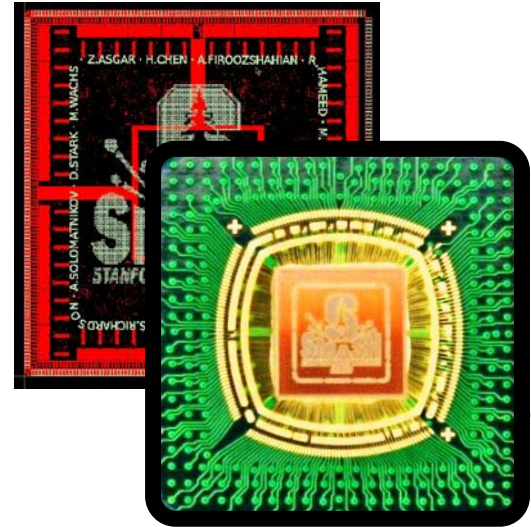
- Fixed configuration of run-time flexible IP blocks





# Why Reconfiguration Is Not Efficient?

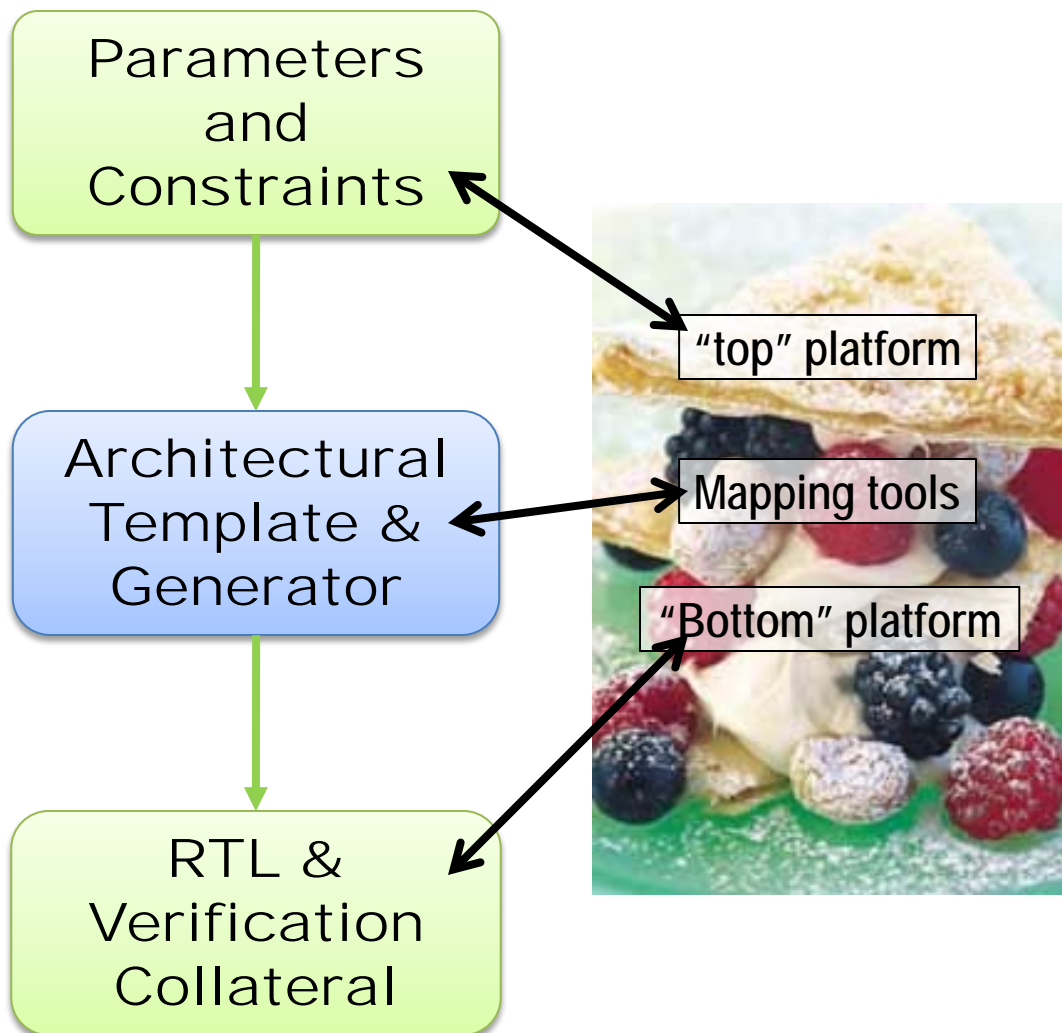
- A reconfigurable system can be modified / tailored to specific application
  - At runtime (i.e., via software)
  - Using the same set of resources
- For example, we created *Stanford's Smart Memories*
  - (Probably) the most reconfigurable CMP ever created
- Fixed architecture w/ mushy (programmable) blocks
  - Does help / amortize the system complexity and reasoning problem!
- Alas... **BIG** problem:
  - The resource-mix is never optimal (un-utilized resources / missing resources)
  - The configuration itself adds inefficiencies (registers, muxes, etc.)



# Need to Rethink Our Ways Again: Move The Configuration Step To Design Time



The right way to create an *Architecture Platform* is not in RTL, but in something that compiles into RTL

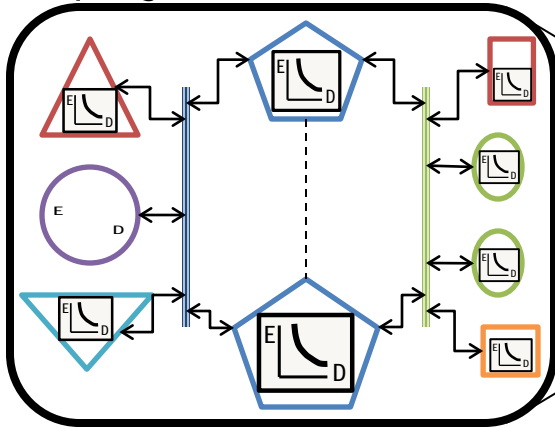


# Need to Rethink Our Ways Again: Move The Configuration Step To Design Time

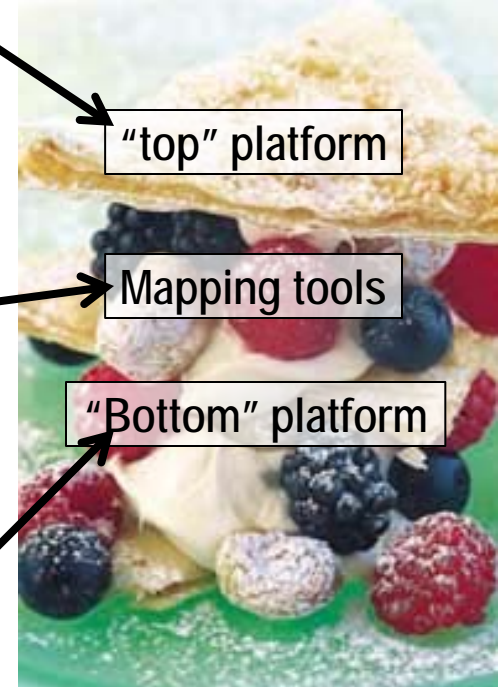
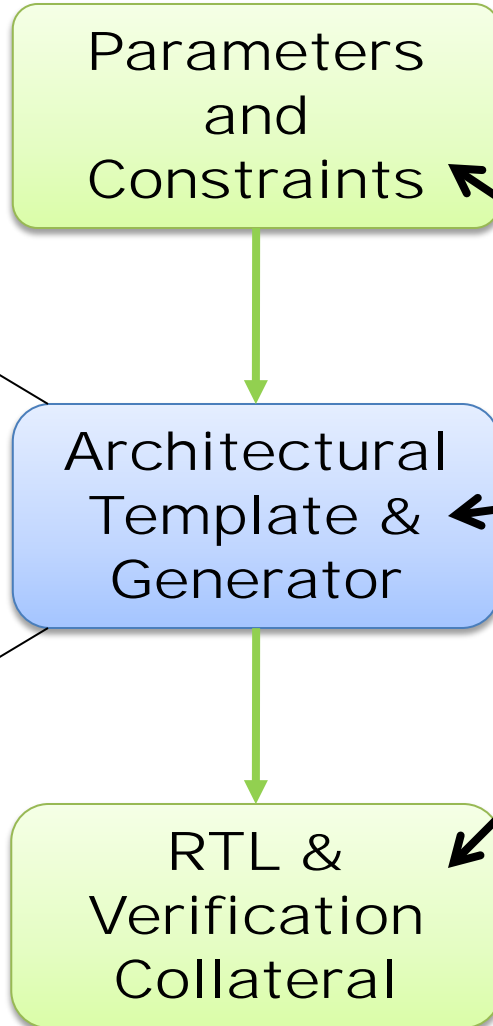


So this is a "virtually" reconfigurable chip

Fixed architecture with mushy (programmable) blocks



Description that encodes system dependencies & trade-offs



"top" platform

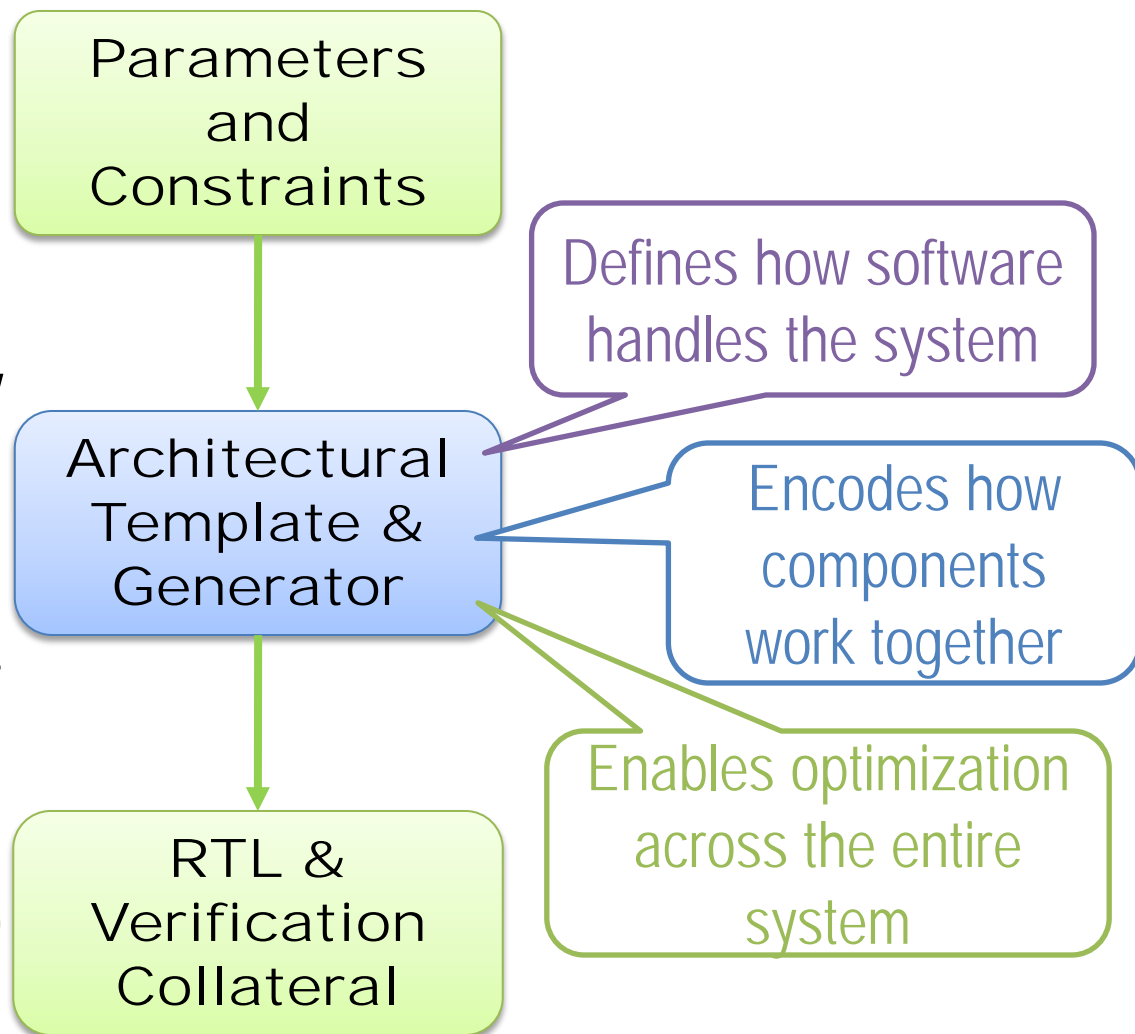
Mapping tools

"Bottom" platform

# Put Our Understanding of The System In A Tool – *Chip Generator*

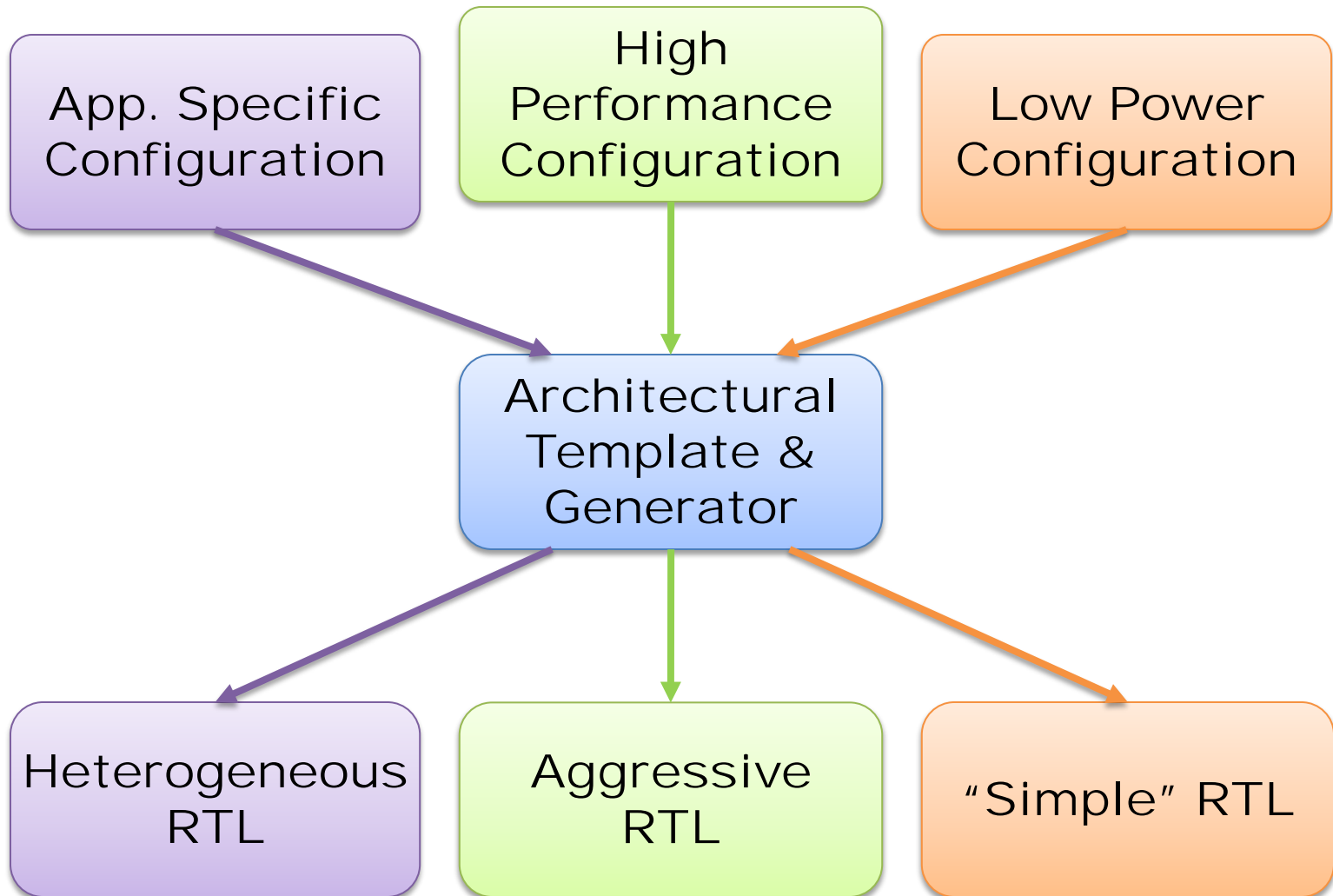


- Unlike logic/high-level synthesis, generators are domain specific
- Each design group likely want to create their own generator
  - But also use other's, lower level, generators
- Need a consistent way for building generators (we'll get back to this a little later)





# One Generator Template Encapsulates Systems





# So We Can Explore The Design Space

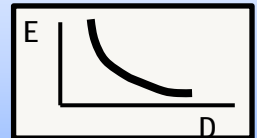
```
private static string HashFunction(string s, int seed) {
    int fieldCount = record.GetFieldCount();
    string[] rowHash = new string[fieldCount];
    for (int i = 1; i <= fieldCount; i++)
    {
        if (record.IsDBNull(i))
        {
            rowHash.Append("null");
        }
        else
        {
            // skip the value of productCode
            if (fieldName == "Product")
            {
                int i = 2;
                rowHash.Append(i + " " + record.GetString(i));
            }
            else if (fieldName == "Seq")
            {
                rowHash.Append(i + " " + record.GetInt32(i));
            }
            else
            {
                rowHash.Append(i + " " + record.GetString(i));
            }
        }
    }
    return rowHash.ToString();
}
```

Architectural  
Description

Energy  
Budget

Simulator

Hardware  
Optimization &  
Generation

Tradeoff  


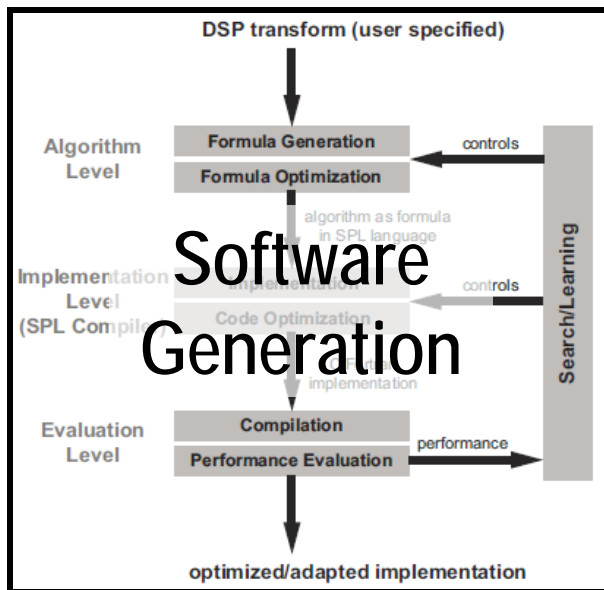
Performance

Power Usage



# Example: CMU's Spiral FFT Generator

- Hardware and software co-optimization framework for DSP applications
  - Given a transformation, Spiral can easily explore the implementation space
  - Final result is optimal hardware and software for a given algorithm + constraints



D'Alberto, Milder et. al., IEEE Comp. Soc., 2007

The screenshot shows the Spiral FFT Generator's configuration interface. It features a table of parameters with columns for parameter, value, range, and explanation. The parameters are categorized into 'Problem specification' and 'Parameters controlling implementation'. The interface includes 'Generate Verilog' and 'Reset' buttons at the bottom.

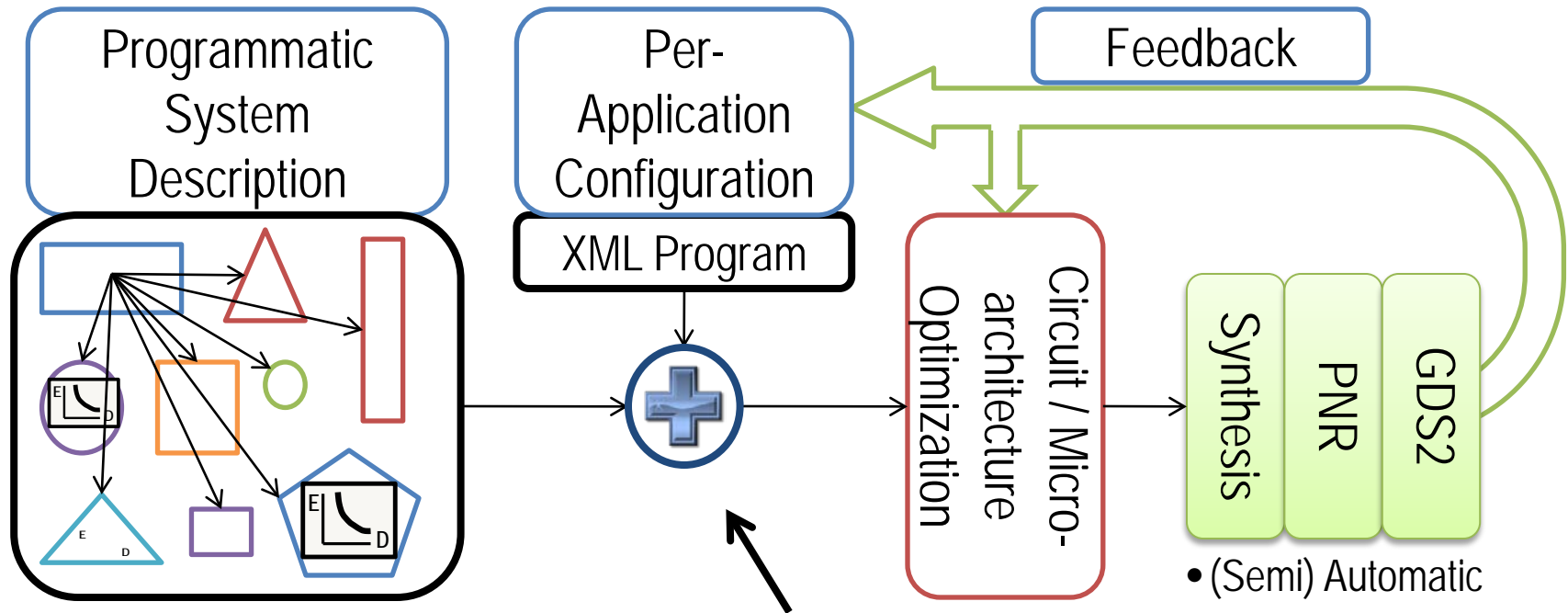
parameter	value	range	explanation
<b>Problem specification</b>			
transform size	64	4-16384	Number of samples (2)
direction	forward		forward or inverse DFT (2)
data type	fixed point		fixed or floating point (2)
	16 bits	4-32 bits	fixed point precision (2)
	unscaled		fixed point scale (2)
<b>Parameters controlling implementation</b>			
architecture	fully streaming		fully streaming (2)
radix	2	2, 4, 16, 32, 64	size of FFT basic block (2)
streaming width	2	2-64	number of complex words per cycle (2)
data ordering	natural in / natural out		natural or digit-reversed data order (2)
BRAM budget	1000		maximum # of BRAMs to utilize (-1 for no limit) (2)

Milder, Franchetti, Hoe and Püschel, DAC, 2008

<http://www.spiral.net/hardware/dftgen.html>



# Generalizing Hardware Optimization & Generation



- Keep flexibility in sub modules
- Encode cross-module dependencies
- Encode validation and software dependencies
- Encode physical / backend dependencies
- Standardize interfacing with configuration tools
- Formal interface for architect / application designer / end client
- Repeat until target reached
- Repeat for different chips



# Standardizing The Creation Of Generators

- We Created *Genesis2*
  - So we (and you!) can build generators
- Premise: Keep it simple – *Genesis2* is just an extension for SystemVerilog
  - Works like a pre-processor (\* but actually is doing a little more)
  - Remove artificial “synthesizability” limitations from the elaboration code
  - Make elaboration explicit; software-like
- Expressive: Encapsulate design trade-offs in blocks (make generators!)
  - Goal is for each block to have a small elaboration program – a “constructor”
  - Enable late, externally driven, tweaking of knobs
- Instead of coding modules, we write programs that produce modules



# Genesis2 Code Snippets

*File: BitReverse.vp*

```

module `mname` (
  //; my $width=parameter(N
  my $assoc = parameter(Name=>'CACHE_ASSOC',
  );
  //; foreach my $idx
  assign data_out[$idx
  data_in[`$width
  //; }

```

*File: OneHotMux.vp*

```

module `mname` (
  GWIDTH', Val=>4);
  JXWIDTH', Val=>4);
  width-1`0),

```

```

input logic [ $mux_width-1 :0] sel,

```

```

my $func = parameter(Name=>'TWID_FUNC',
  List=>['cos', 'sin'],
  Val=>'cos');

```

```

my $width = parameter(Name=>'BIT_WIDTH',
  Min=>4, Max=>16, Step=>2
  Val=>6);

```

```

endcase

```

```

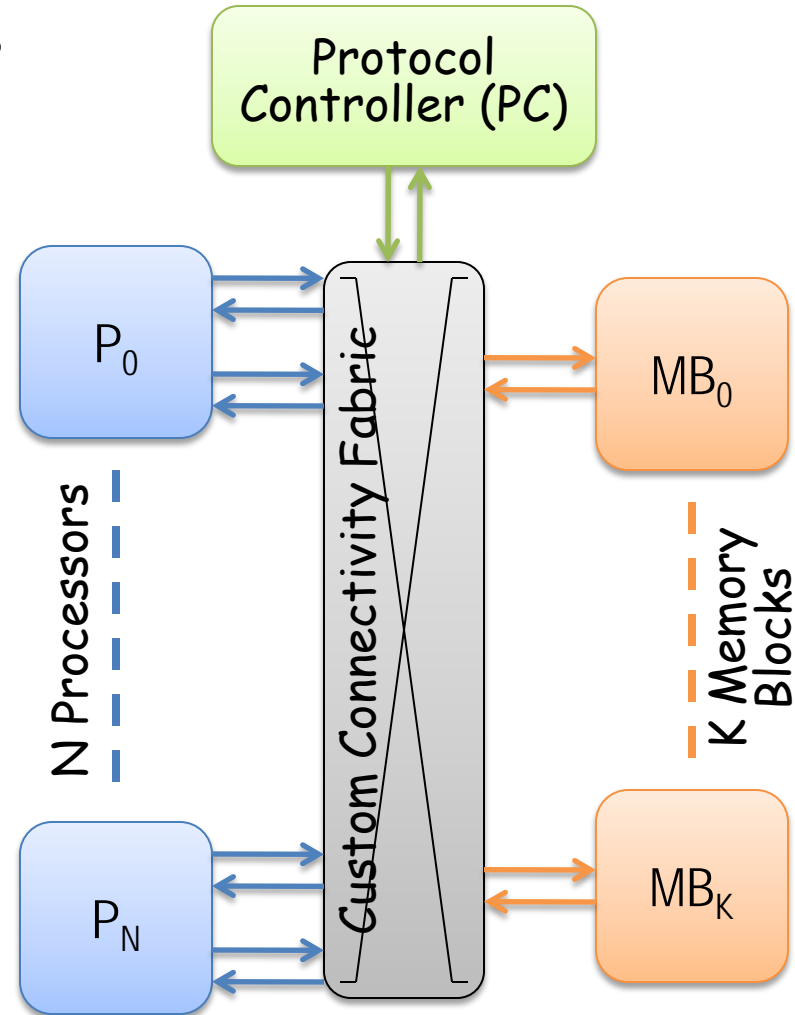
my $init_arr= parameter(Name=>'ROM_CONTENT',
  Val=>[ ]); Initial param value
  is an empty list!

```



# Example: Our Chip Multiprocessor Generator

- What would adding a processor require?
  - More MB's; Bigger PC; More fabric
- Changing a proc (e.g., scalar to vliw)?
  - Change the word width of the bus and the relevant memory block
- What about changing the protocol? Or adding a new memory operation?
- Conclusion:
  - Meaningful changes are not just local. They require system-wide modifications
  - Need to encode the *system* reasoning.



Work done with Andrew Danowitz and Megan Wachs



# Grow Beyond The Per-Module Scope

- **Hierarchical: Build bigger generators out of smaller generators**
  - Remember: each generator is the encoding of its designer's thought process and the encapsulation of many design trade-offs
  - So instantiate the generator; Not just one instance of "cemented" Verilog
- **Scope: Open the system scope to the individual generators**
  - To resolve structural constraints between modules

```
my $CacheObj = generate('Cache', 'Dcache',  
                        PROCESSOR => $ProcObj);
```

Sub-Generator name → Instance name → pointer to relevant processor

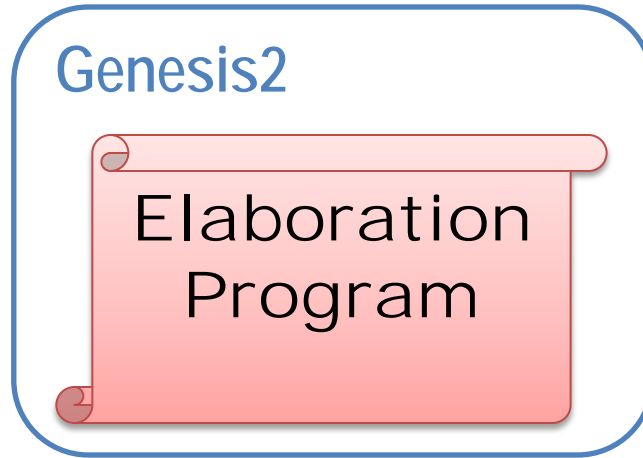
- **Still need to control the internal knobs of the cache from the outside world**
  - E.g., way size, associativity, etc.



# Standardized Parameter I/O Through XML

```
<cg_cmp>
<Parameters>
<num_cpus>2</num_cpus>
</Parameters>
<SubInstances>
<p1>
<Parameters>
<type>risc</type>
</Parameters>
<SubInstances> ...
</SubInstances>
</p1>
<p2>
<Parameters>
<type>vliw</type>
</Parameters>
<SubInstances> ...
</SubInstances>
</p2>
</SubInstances>
</cg_cmp>
```

XML  
Input

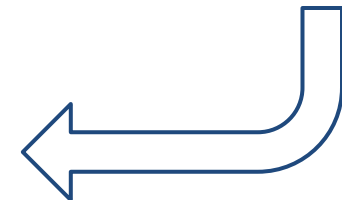


```
<cg_cmp>
<Parameters>
<num_cpus>2</num_cpus>
</Parameters>
<SubInstances>
<p1>
<Parameters>
<type>risc</type>
</Parameters>
<SubInstances> ...
</SubInstances>
</p1>
<p2>
<Parameters>
<type>vliw</type>
</Parameters>
<SubInstances> ...
</SubInstances>
</p2>
</SubInstances>
</cg_cmp>
```

XML  
Output



Refinement of the architectural parameters by application designer and/or optimization tools





# Standard Graphical User Interface

top

tst2dut_cfg_ifc	dut2tst_cfg_ifc
DUT	tb

**Parameters for instance "top"**

User-tweakable parameters:

MODE	<input type="text" value="VERIF"/>	<i>This is the mode of the generation. Also, it's a VERY LONG COMMENT....</i>
ASSERTION	<input type="text" value="ON"/>	<i>This is the assertion mode of the generation.</i>
QUAD_ID	<input type="text" value="0"/>	<i>In this example, can have up to 64 quads(!)</i>
TILE_ID	<input type="text" value="0"/>	<i>Any number greater than 0, for illustration.</i>
NUM_PROCESSOR	<input type="text" value="1"/>	<i>no comment</i>
NUM_MEM_MATS	<input type="text" value="1"/>	<i>no comment</i>

Immutable parameters:

Work done with Steve Richardson, Megan Wachs, and Andrew Danowitz



# Genesis2 Now Also Used For CMU Generators

The screenshot shows a web browser window with the URL `genesis.web.cmu.edu/g/mydesign-11576.php`. The page title is "Parameters for instance 'top.topcontrol\_eg.backprojection'". The interface is divided into two main sections: "Tweakable parameters" and "Immutable parameters".

**Tweakable parameters:**

TRIPRECISION_LOG2	16
INTER_RESOLUTION_LOG2	6
ROI_startx	0
ROI_starty	0
ROI_endx	31
ROI_endy	31
ROI_height	32
ROI_width	32

**Immutable parameters:**

IMAGEWIDTH_LOG2	5
RADONSIZE_LOG2	6
IMAGEHEIGHT_LOG2	5
RADONSIZE_ORIG	49
PIXELPRECISION_LOG2	16
ANGLESIZE_LOG2	6

At the bottom right of the parameter list is a "Submit changes" button. Below the parameter list, there is a link: "Download tar of current design (.vp and .xml files) (must first 'Submit changes')".

On the left side of the browser window, there is a sidebar with the instance name "top.topcontrol\_eg.backprojection" and navigation buttons "<", ">", and "UP". Below this, there are four input fields: "twi\_inter\_cos", "twi\_inte\_sin", "linear\_inter", and "destmemory".

Designed by Qiling Zhu, Carnegie Mellon University



# Summary / Conclusions

- Power is today's greatest design constrain → Calls for customization
- Costs of creating new systems prohibitive → Due to system complexity
- *Chip Generators* encapsulate the designer's understanding of the system, as well as the design trade-offs
  - Constrained architecture built off (design-time) flexible blocks
  - The artifact produce is the process of making a chip, not a design instance
- *Genesis2* is a simple extension to SystemVerilog that enables and standardize the creation of hierarchical chip generators

*Genesis2* can be downloaded from <http://genesis2.stanford.edu/>



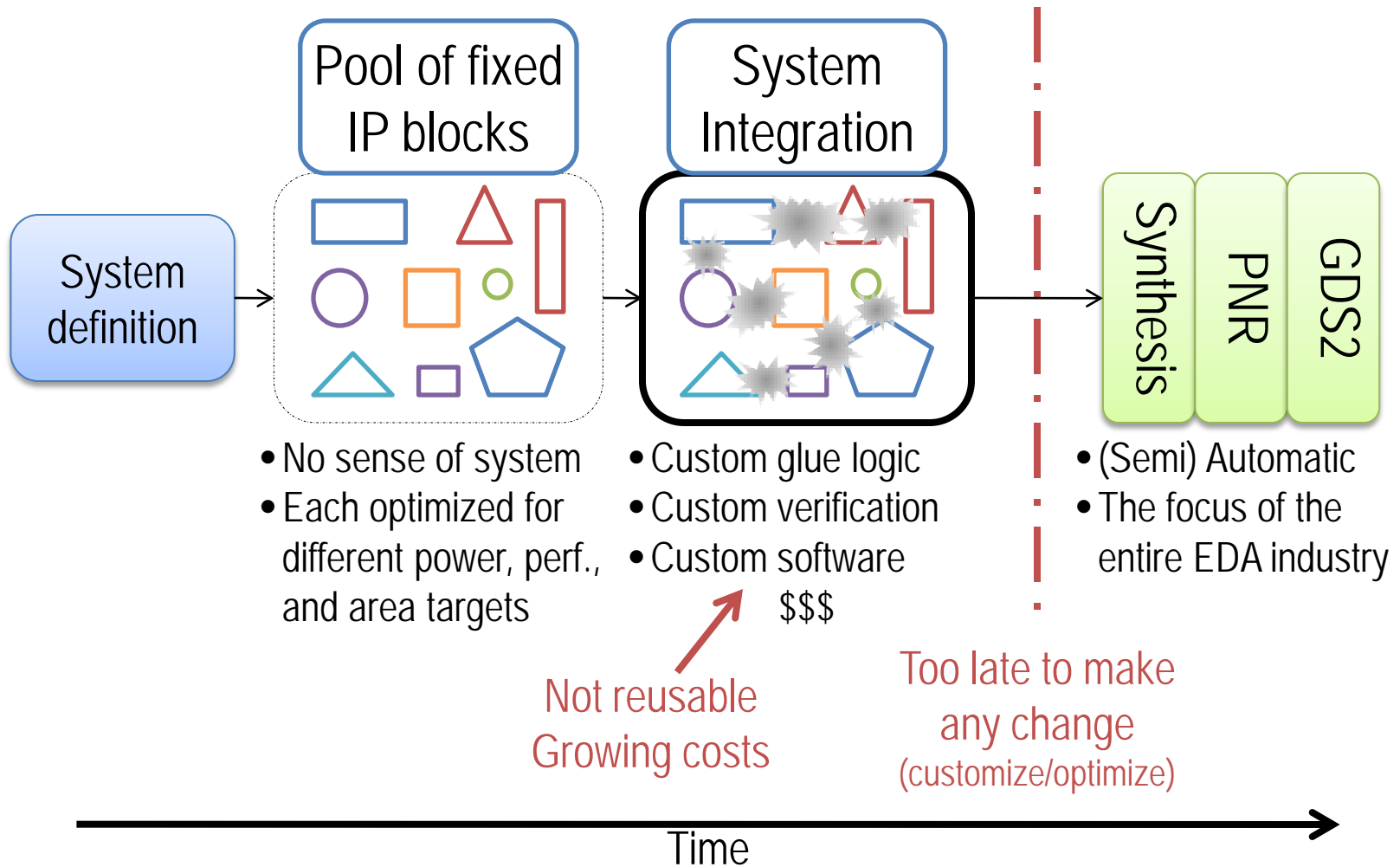
**THANK YOU!**



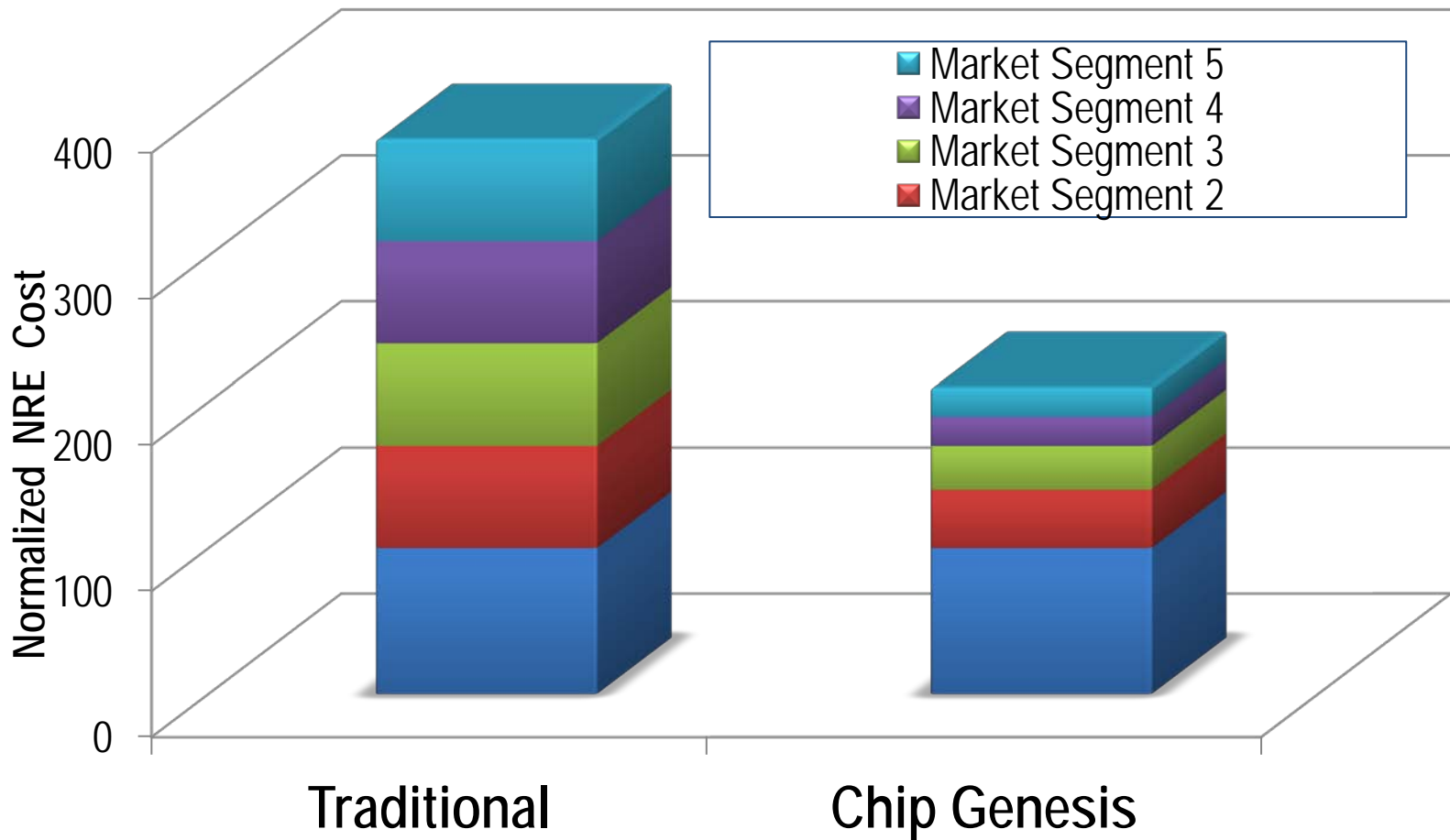
# BACKUP SLIDES



# Why SoC Costs Are Out Of Control?



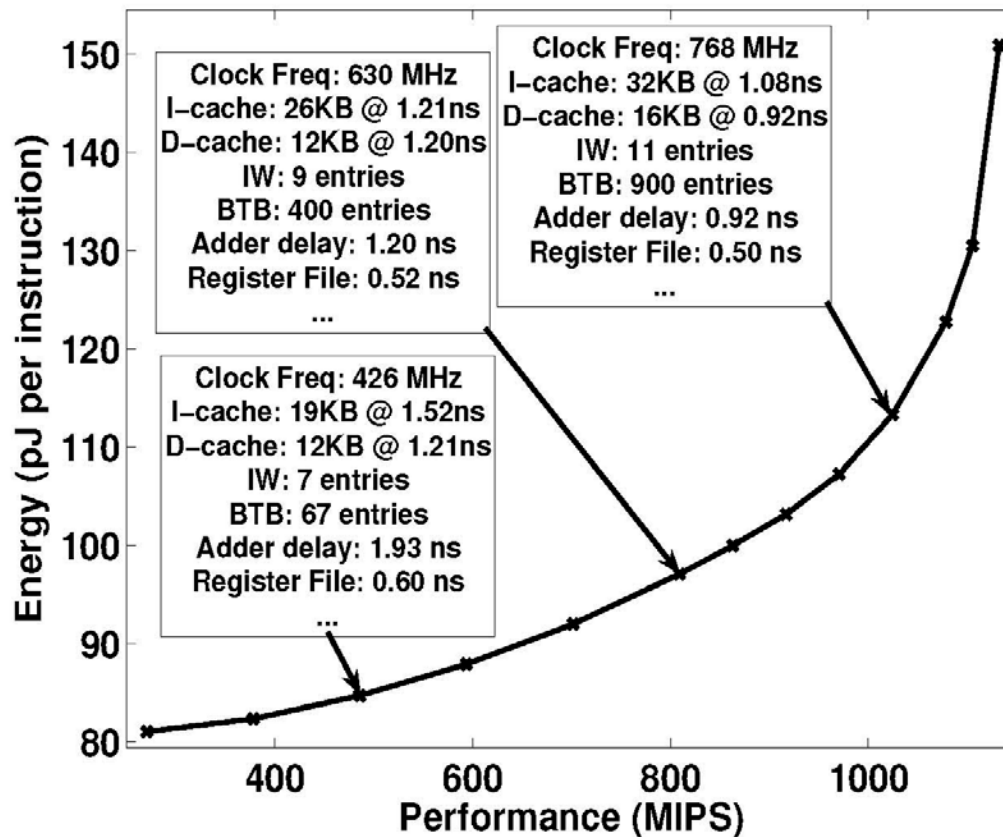
# End Result 1: Better Amortized Cost Structure



# End Result 2: Optimization At The System Level

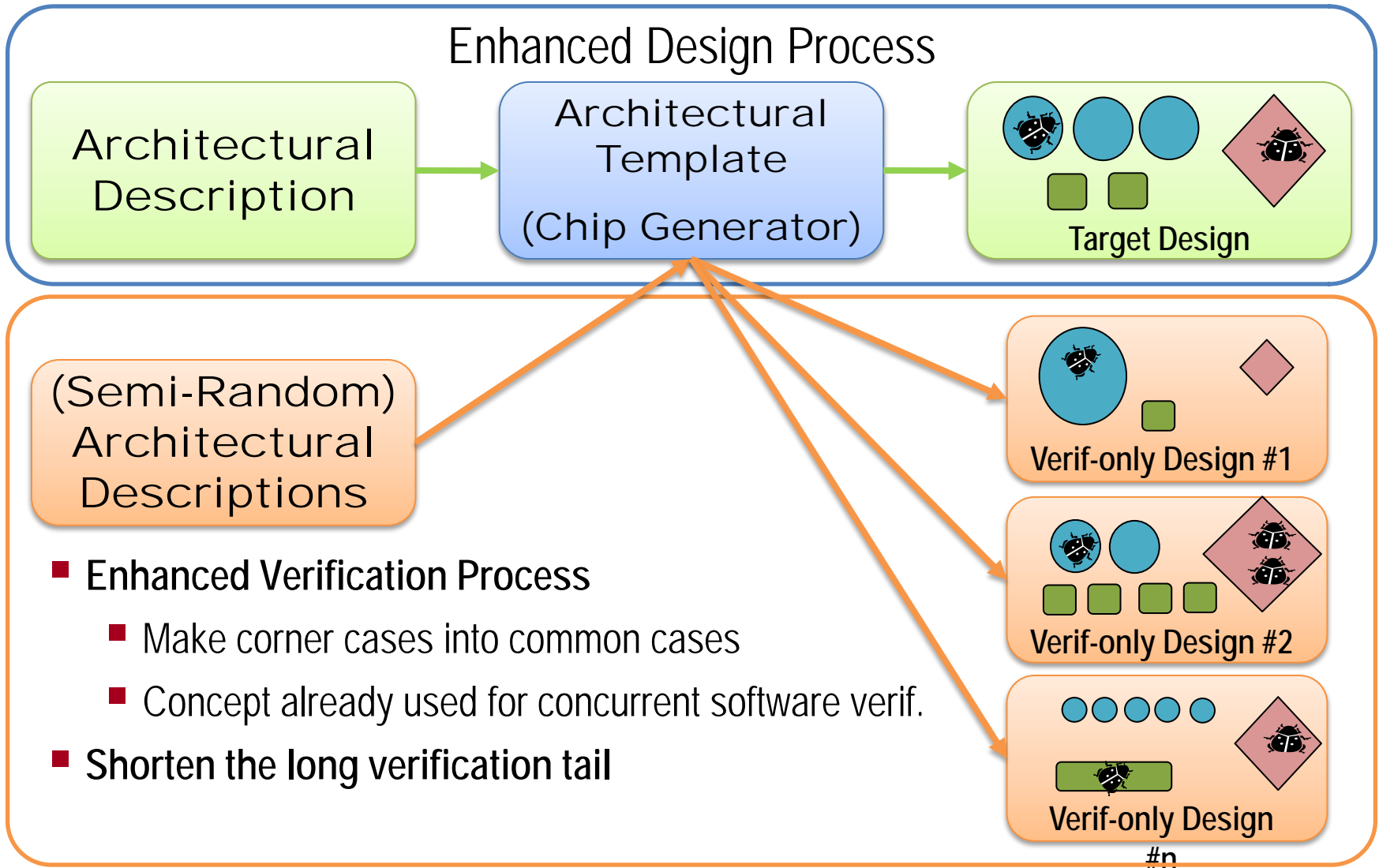


- Example: Best processor architecture configuration at various power / performance budgets



Source: Omid J. Azizi,  
PhD Thesis, Stanford 2010

# End Result 3: Verification Team Also Leverages The Generator

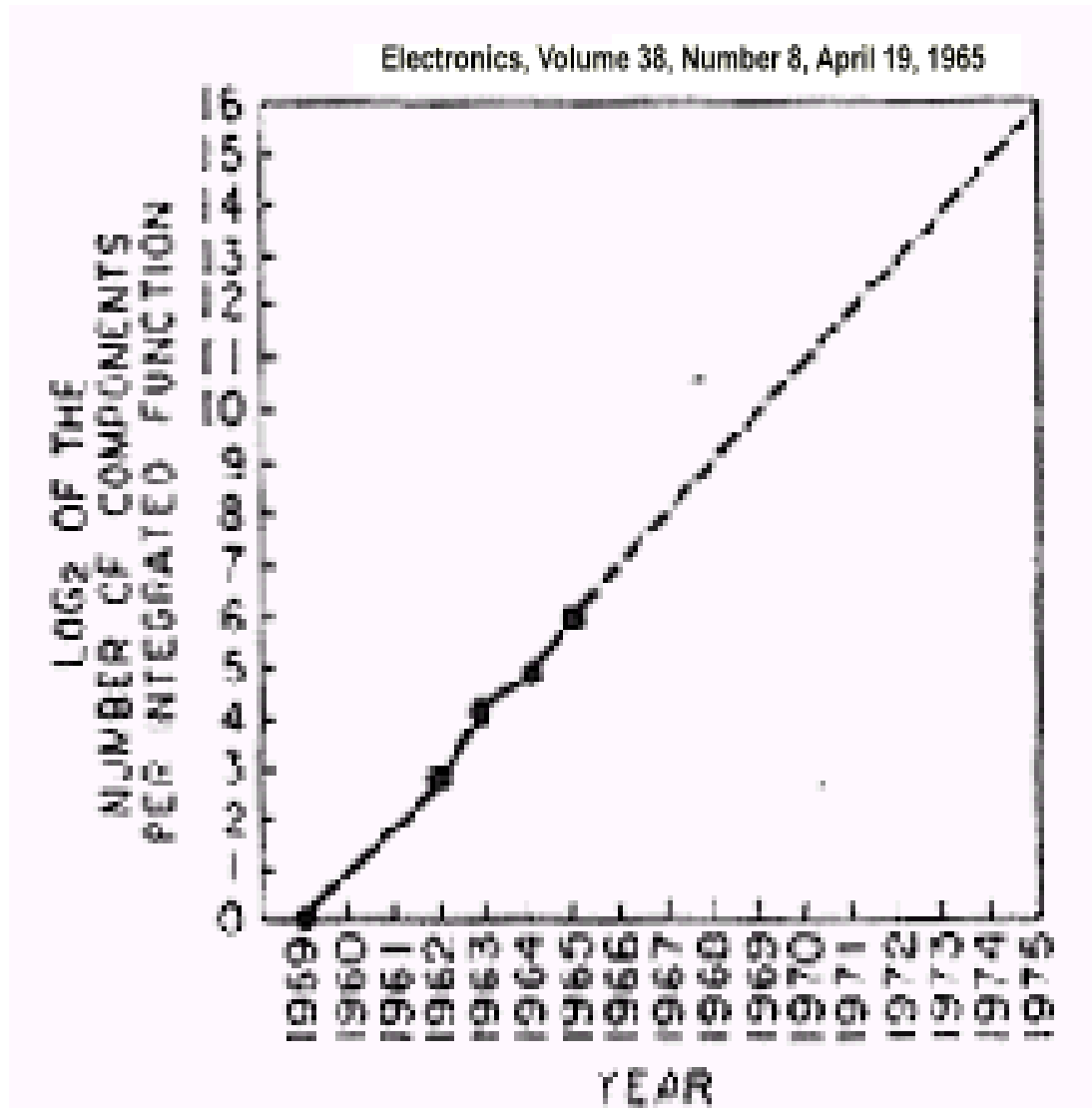




# THE END OF DENNARD SCALING

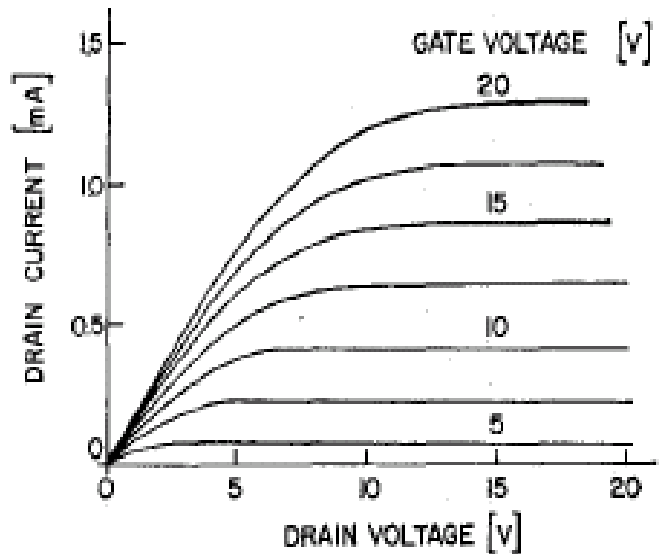


# Historical Technology Scaling

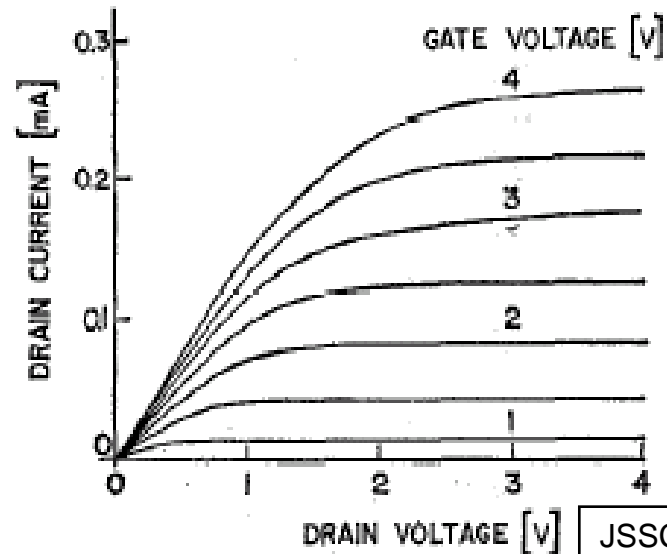




# Dennard's MOS Scaling (1974)



$t_{ox} = 1000 \text{ \AA}$   
 $L = W = 5 \mu\text{m}$   
 $V_{sub} = -7\text{V}$   
 $\psi_s = 0.65\text{V}$



$t'_{ox} = 200 \text{ \AA}$   
 $L' = W' = 1 \mu\text{m}$   
 $V'_{sub} = -1\text{V}$   
 $\psi'_s = 0.73\text{V}$

JSSC Oct 74, pg 256

- In this ideal scaling ( $\alpha < 1$ )
  - $V$  scales to  $\alpha V$ ,  $L$  scales to  $\alpha L$ ,  $W$  scales to  $\alpha W$
  - So:  $C$  scales to  $\alpha C$ ,
  - $E = V/L$  is constant, so  $i$  scales to  $\alpha i$  ( $i/m$  is stable)



# The Triple Play of Historical Scaling

- Over three decades of constant field (Dennard) scaling
  - $V$  scales to  $\alpha V$ ,  $L$  scales to  $\alpha L$ ,  $W$  scales to  $\alpha W$  ( $\alpha < 1$ )
- Everybody wins:
  - Get more transistors, more gates  $1/\alpha^2$
  - Gates get faster, delay scales as  $\alpha$
  - Energy per switch is reduced  $\alpha^3$
- For the same power and area as the previous design
  - Compute grows as  $1/\alpha^3$
- Architects use this to improve computer performance



# RANDOM BACKUP SLIDES



# Turn The Design Process Inside Out

- **Conventional System-on-Chip Design:**
  - Designer creates system from complex components
    - IP components are designed in advance
    - End with a new connection of fixed components
  - SoC designer has to connect multiple IP blocks:
    - Interface adapters between different blocks
    - **Complex verification**
- **Chip Generator:**
  - Designer tunes parts in a “fixed” system architecture
    - Fixed design of squashy components
  - Functional interfaces remain constant
    - Reusable validation

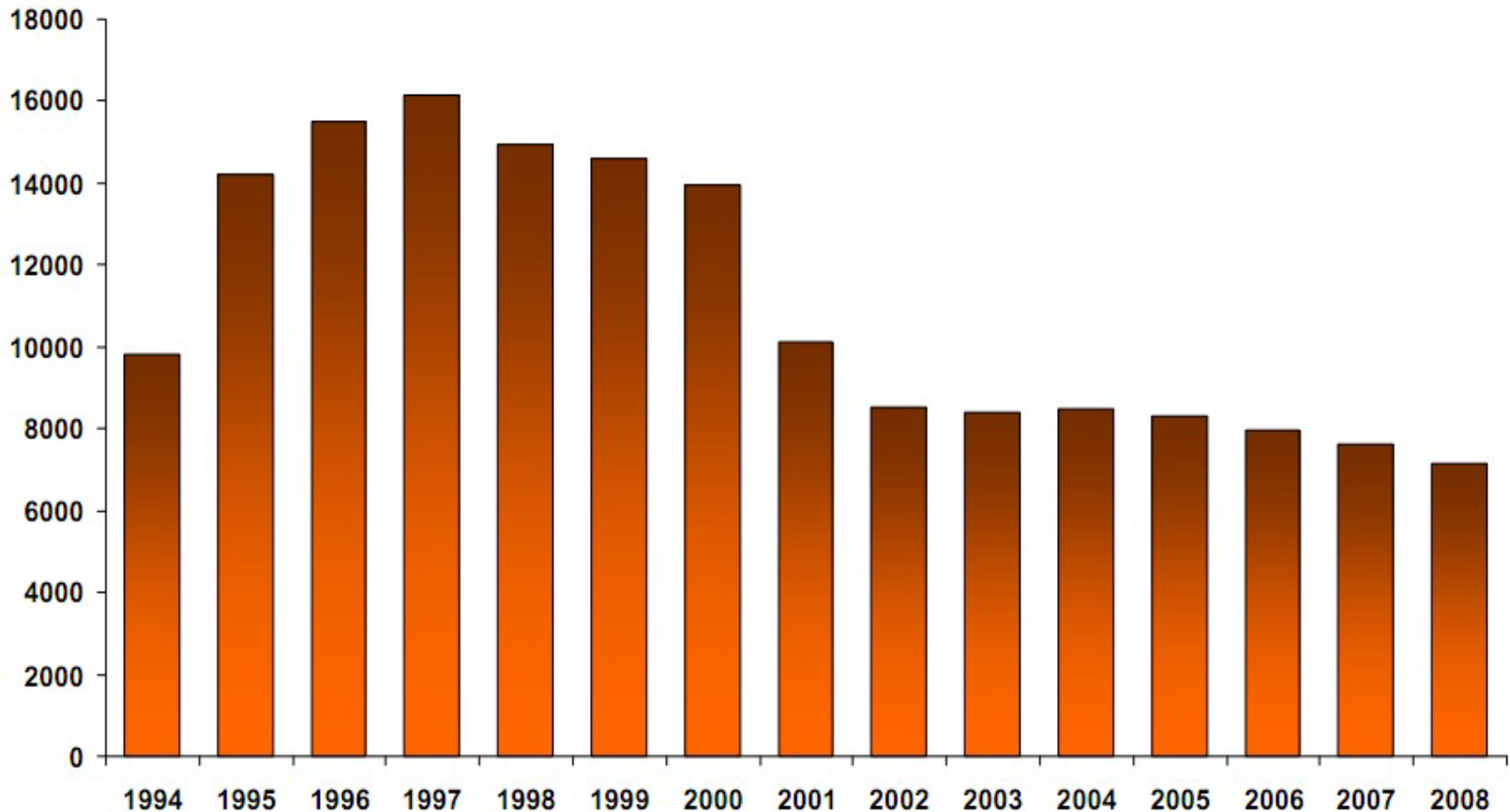


# Silicon Compilers Again?

- **Well, yes and no**
  - No: I don't expect it will take application code and "compile" it
  - Yes: Each time we create a generator we essentially define a "language" that accepts an architectural "description," and "compiles" it to silicon
- **Designers today encode results; We don't encode our knowledge**
  - We think of many alternatives in a design; then choose one
    - For the next application perhaps another alternative is best
  - We optimize designs at each level, but then freeze them
    - What happens if we let the design remain flexible?
- **Instead, embed explicit elaboration instructions into modules**
  - Same function that constructors provide for classes



# Semiconductors Logic Design Starts



Source: Gartner/Dataquest Market Trends

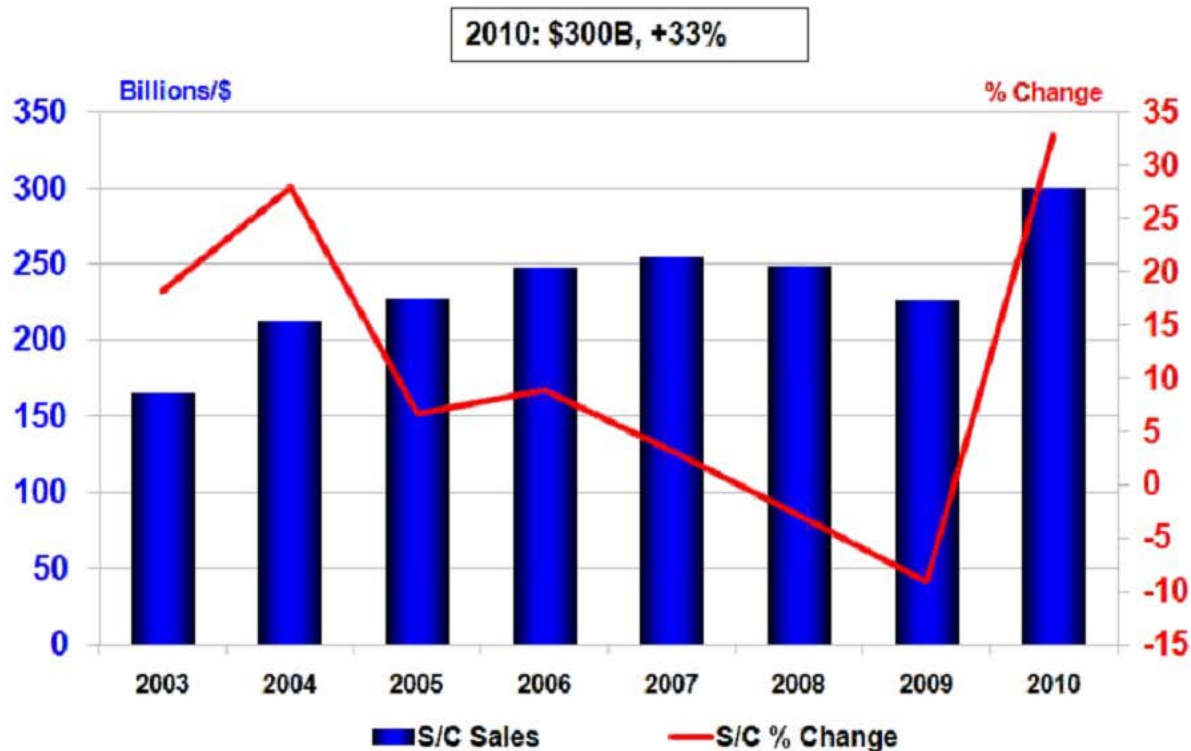
\* Without programmable logic

Compiled by Walden C. Rhines, CEO, Mento Graphics



# Semiconductor Market Size

## Industry Revenue Growth: SIA Forecast

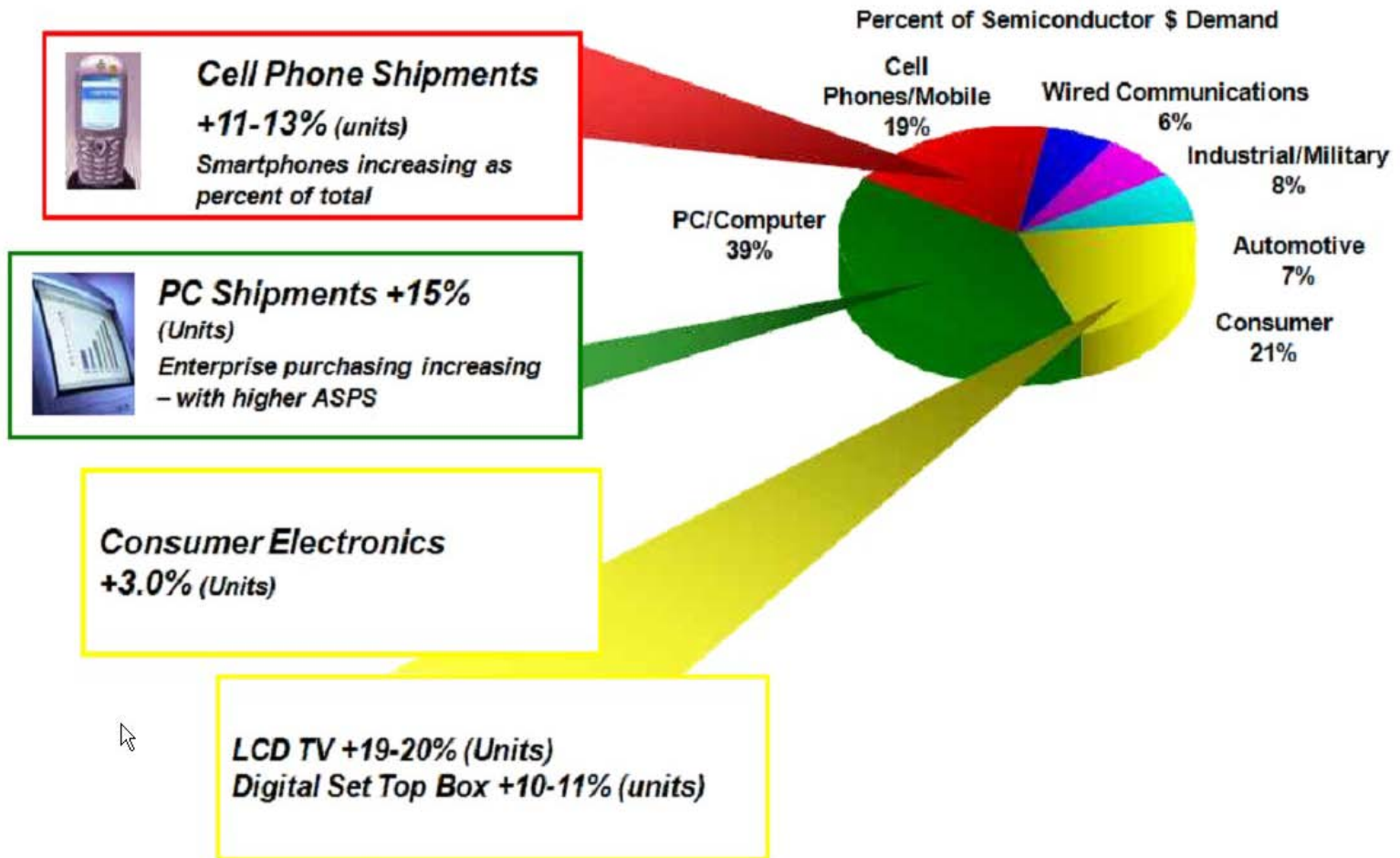


Source: SIA November 2010 Forecast





# Semiconductor Market Breakdown (2010)



Sources: SIA June 2010 Forecast/Credit Suisse/J.P. Morgan/Suppli  
Note: Military is <1% and is included in Industrial.