Automated Synthesis and Verification of Embedded Systems: Wishful Thinking or Reality?

Wolfgang Rosenstiel

Computer Engineering
Wilhelm-Schickard-Institute for Informatics
University of Tuebingen
Germany
From Hardware Supply to the Final Application

Sphere of design influence

Source: Medea
Embedded Software is Becoming More Valuable

Contribution to revenue, %

<table>
<thead>
<tr>
<th>Year</th>
<th>Electronics Software</th>
<th>Electronics Hardware</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>80</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>2015</td>
<td>60</td>
<td>25</td>
<td>15</td>
</tr>
</tbody>
</table>

Innovation by Source, %

<table>
<thead>
<tr>
<th>Year</th>
<th>Electronics Software</th>
<th>Electronics Hardware</th>
<th>Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>70</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>2015</td>
<td>80</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Source: 2003 McKinsey-PTW HAWK survey (Institute for Production Management at the Technical University at Darmstadt, McKinsey analysis)
The Automotive Silicon Drive

Cars:ECU:S/C = 1:5:12

Percent of Growth (2000 = 100%)

Semiconductors 580%
ECU 300%
Cars & light trucks 140%

Source: Bosch
Added Value Through New Electronic Functions

Source: Mercer, a.o.
The Growing Role of Chip Makers

- The electronic part of final applications is increasingly determined and designed by semiconductor chip makers.
- Customer interface is shifting to the boundary between the embedded code in the semiconductor device and the application software.
- Building intelligent systems for the future requires close collaboration between the innovation teams of chip suppliers, chip customers.

→ EDA is the bridge between them.

Source: Panel Discussion, edaForum07
Outline

- EDA challenges
- System Level Design
- Verification
- Synthesis
- Conclusions
Design Costs

Source: ITRS 2008 based on data from Gary Smith

Design cost (SM)

Total HW Engineering Costs + EDA Tool Costs
Total SW Engineering Costs + ESDA Tool Costs

Automotive Electronics

- Today's readmission rate for cars is ~ 70 000 000 cars per year
- Average of 20 to 70 embedded systems per car
- 50% development effort spend on software engineering
- Every 2nd car recall caused by software problems

→ Software quality assessment plays an important role in cost reduction and customer satisfaction
Verification Problems

- **2004**
  - Pontiac recalls the Grand Prix since the software did **not understand leap years**. 2004 was a leap year.

- **2003**
  - A BMW 520 trapped a Thai politician when the computer crashed. The **door locks**, **windows**, **A/C and more were inoperable**. Windshield was smashed to get him out.

- **2002**
  - BMW recalls the 745i since the **fuel pump would shut off if the tank was less than 1/3 full**.
  
Verification Challenges

- **Challenge: Complexity growth**
  - Example: Diesel systems
  - 2007
    - Calibration parameters: 16 k
    - Performance: 300 MIPS
    - Memory: 4 MByte

- **Software coding errors**
  - Finite-state machines
  - Timing
  - Stack/memory overflows
  - Consistencies
    - Non-volatile memory

Source: *Software Bugs seen from an Industrial Perspective. CAV07* [Kropf07]
Outline

- EDA challenges
- System Level Design
- Verification
- Synthesis
- Examples and Results
Design Flow of Distributed Embedded Systems

- Model based design of distributed systems
  - platform dependent development of the application software (UML, Matlab/Simulink, C++)
  - early consideration of the planned target platform in the model based system design (UML)
  - mapping of function blocks on architecture components
  - use of virtual prototypes for the abstract modeling of the target platform

- Early analysis of the system integration
  - early verification based on virtual prototypes
  - formal and semiformal software verification

- Seamless transition to the real prototype
  - automatic “target code” generation
  - automatic high level synthesis
  - co-simulation/emulation
  - SystemC as intermediate representation
Challenge 1. Move from document-centric to model-centric.

Challenge 2. Interfaces between requirements, specification, verification aims, and implementation.

Challenge 3. Tracing of requirements and verification aims.

Challenge 4. Automatic generation of software, hardware, and test cases (ATPG).
Design and Verification Process with UML

- Interfaces between specification, verification aims, and implementation
- Enable code generation
- Enable test case description
- Enable linking and tracing of design components
- Missing \(\rightarrow\) requirements
- Missing \(\rightarrow\) support the verification process
System Architecture

UML 2.x Diagram

Structure Diagram

Class Diagram
Composition Diagram
Object Diagram

Component Diagram
Package Diagram
Deployment Diagram

Behavior Diagram

Activity Diagram
State Machine Diagram
Use-Case Diagram

Interaction Diagram

Communication Diagram
Sequence Diagram
Timing Diagram
Interaction Overview Diagram
System Behavior

UML 2.x Diagram

Structure Diagram

Class Diagram
Composition Diagram
Object Diagram

Component Diagram
Package Diagram
Deployment Diagram

Activity Diagram

Behavior Diagram

State Machine Diagram
Use-Case Diagram

Interaction Diagram

Communication Diagram
Sequence Diagram
Timing Diagram
Interaction Overview Diagram
Scenario / Test case
SysML

- Structure Diagram
  - Component Diagram
  - Package Diagram
  - Deployment Diagram
  - Parametric Diagram

- Requirements-Diagram
  - Activity Diagram

- Behavior Diagram
  - State Machine Diagram
  - Use-Case-Diagram

- Interaction Diagram
  - Communication Diagram
  - Sequence Diagram
  - Timing Diagram
  - Interaction Overview Diagram

New diagrams in SysML
Extended/modified diagrams in SysML
Challenge

- System analysis of UML/SysML diagrams
- Generating for verification aims (Assertions)
Outline

- EDA challenges
- System Level Design
- Verification
- Synthesis
- Conclusion
System Analysis through Assertions

System Specification (UML/SysML)
- Sequence Diagram
- Deployment Diagram
- Structure Diagram
- Communication
- Prozess to Module
- System-Architecture

Abstracte Property

Assertions

Simulation
Formale Verification
Test pattern indicate test quality
- Assertions improve error localization
- Assertions for functional and formal verification
Software Modeling

- Integers, arrays, floating point
- Arithmetic operations: +, -, *, /, %
- Pointers
- Procedures

- Formal model
  - Static and control-flow operations
  - Model here means a FSM
  - Boolean variables

- Simulation model
  - Dynamic aspects (e.g., dynamic allocation)
  - Data-flow arithmetic operations (e.g., multiplication and division)
Outline

- EDA challenges
- System Level Design
- Verification
  - Semi-Formal Verification Flow and Results
- Synthesis
- Conclusion
Embedded Software Modeling: Overview

Complex C program

int p;
int main() {
    int i = 0;
    while( i != p) {
        i = i + 1;
    }
    return(0);
}

„Simple“ C program

CFA Generator

Control flow automata

CIL

1) CIL: http://cil.sourceforge.net

Pred (i== p)

Block(i=0)

Pred (i != p)

Block (i = i + 1)

READ → F EEE_OK

Properties / Critical states

Formal verification

SymC

Model Generator

C/SystemC

Simulation model

Simulation

Formal model
module main
signal s3, s2, s1, i3, i2, i1, ret : boolean;
input p2, p1 : boolean;

init
s3 == false; s2 == false; s1 == false;
define
state0 := (!s3 & !s2 & !s1); state1 := (!s3 & !s2 & s1);
state2 := (!s3 & s2 & !s1); state3 := (!s3 & s2 & s1);
state4 := (s3 & !s2 & !s1);
eqCheck := ((p2&i2)|(!p2&!i2))&((p1&i1)|(!p1&!i1));
...
trans
next (s3) == state2; next (s2) == state1;
next (s1) == state0|(state1 & !eqCheck)|state3;
next (i3) == (false & state0) | (m1.c2 & state3) | (!state0 | state3) & i3;
next (i2) == (false & state0) | (m1.s1 & state3) | (!state0 | state3) & i2;
next (i1) == (false & state0) | (m0.s0 & state3) | (!state0 | state3) & i1;
next (ret) == (false & state2) | (!state2 & ret);
...
invar
add0.a0 == (state3 & i1); add0.b0 == (state3 & true);
add1.a1 == (state3 & i2); add1.b1 == (state3 & false);
...
end
Temporal Checker Framework

- Property specification in SystemC models
  - Proposition class
- Property synthesis and checking
  - LTL with Time Bound
    - F[2] a
- Customizable actions in special states

- SystemC model
- PSL / FLTL properties
- AR
- SystemC kernel & temporal checker library
- Customizable actions: assertion messages, exceptions
Experimental Results

- EEPROM Emulation Software
- Derived 86 Properties from EEELib specification
- Performed 93 BLAST-runs (timeout set to 24h)
  - Total verification time: 532 hours (~22 days)
- Results:
  - 7 trials resulted in BLAST internal error
  - 8 trials did not finish within 24h
  - 78 trials resulted in “the system is safe” message
### Experimental Results

- **NVM-Software**

<table>
<thead>
<tr>
<th>Properties</th>
<th>Results CBMC</th>
<th>Verif. Time CBMC</th>
<th>Results BLAST</th>
<th>Verif. Time BLAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The index of data blocks requested should be valid between 0 and the maximal number of data blocks or be assigned to FAILED if the given block is invalid.</td>
<td>No error</td>
<td>2.34s</td>
<td>No error</td>
<td>&lt; 1min.</td>
</tr>
<tr>
<td>2. A data block is only initialized when the address of external buffer is assigned.</td>
<td>Internal error</td>
<td>50 min.</td>
<td>Internal error</td>
<td>34 min.</td>
</tr>
<tr>
<td>3. A request has only two state: &quot;NO_REQUEST&quot; or &quot;WRITE_REQUEST&quot;.</td>
<td>No error</td>
<td>2.38s</td>
<td>Error</td>
<td>&lt; 1min.</td>
</tr>
<tr>
<td>4. If a block is a double buffered block, NVM system should output 1 or 0 to identify the valid block.</td>
<td>Error</td>
<td>35.58s</td>
<td>No error</td>
<td>&lt; 1min.</td>
</tr>
</tbody>
</table>
Outline

- EDA challenges
- System Level Design
- Verification
- Synthesis
- Conclusions
First generation of synthesis tools were not accepted...

One wait statement per loop

```java
for (int i = 0; i<4; i++) {
  wait();
  e = (a * c * d + i);
}
```

At least one wait statement between two write operations the same memory port

```java
e = (a * c * d);
for (int i = 0; i<4; i++) {
  wait();
  real_out.write(e + i);
  // Error: no wait
  real_out.write(e + i + i);
}
wait();
```
... first generation of synthesis tools were not accepted...

```java
if (a < b) {
    e = (a * c * d);
    wait();
}
else if (a = b) {
    e = (b * c * d);
    // Error: no wait
} // Error: ELSE path not considered

if (a < b) {
    e = (a * c * d);
    wait();
} else if (a = b) {
    e = (b * c * d);
    wait();
} else {
    wait();
}
```

// Initialisierung
out_valid.write(true);
out1.write(0);
out2.write("11111111");
// Error: no wait
while (true) {
    ...//process behavior
}

// Initialisierung
out_valid.write(true);
out1.write(0);
out2.write("11111111");
wait();
while (true) {
    ...//process behavior
}
```
...first generation of synthesis tools were not accepted...

```cpp
for (int i = 0; i<4; i++) {
    e = (a * c * d + i);
    real_out.write(e);
    // Error: no wait
    if (i == 3) break;
    wait();
}
```

```cpp
for (int i = 0; i<4; i++) {
    e = (a * c * d + i);
    real_out.write(e);
    wait(); // shift wait in front of if
    if (i == 3) break;
}
```

put $n$ waits between data dependent I/O read and I/O writes if the included calculation requires $n$ cycles

```cpp
ie = (a * c * d);
real_out.write(e);
// Error: no wait
for (int i = 0; i<4; i++) {
    real_out.write(e + i);
    wait();
    c = data_in.read();
    e = (a * c * d);
}
```

```cpp
e = (a * c * d);
real_out.write(e);
wait();
for (int i = 0; i<4; i++) {
    real_out.write(e + i);
    wait();
    c = data_in.read();
    e = (a * c * d);
}
```

put $n + 1$ waits after the loop condition and $n + 1$ waits after the end of the loop, if $n$ cycles are necessary to calculate the loop condition...
"Cycle-Fixed" guidelines...

```c
for (int i = 0; i<4; i++) {
    e = (a * c * d + i);
    wait();
}
```

```c
for (int i = 0; i<4; i++) {
    e = (a * c * d); // Error: no wait;
    wait();
    for (int i = 0; i<4; i++) {
        e = (a * c * d + i);
        wait();
    }
    // Error: no wait;
}
```
High Level Synthesis is getting more acceptance and attention

- Known customers: ST, Qualcomm, Alcatel, Fujitsu, Panasonic, Toshiba, Thales, Sanyo, Ericsson, Pioneer, Motorola, Micronas, NXP, Broadcom, Sony, Canon
- More and more commercial tools:
- More than 50 ASIC Tape-outs reported in 2007, probably more in 2008

(Older) market analysis from Gary Smith:
Mentor Graphics – CatapultC

- **Optimizations**
  - Bus and memory architecture
  - Datapath micro-architecture
  - Power-gating
  - Multiple clock domains
  - Voltage scaling for dynamic power
  - Multi-threshold for static power
  - Clock gating
  - Pipelining

- **Verification**
  - Automated generation of SystemC transactors
  - Transactors convert function calls to pin-level signal activity

- **Interface Synthesis**
- Technology is characterized for accurate timing and area
  - Operating conditions, voltage thresholds, operator arch, speed grade, etc

- Hardware is optimized using these technology libraries
  - Like having synthesis timing guru creating your RTL

- Supports IP, DesignWare, Custom Interfaces, FPGA macros

- Enables optimum ASIC and FPGA synthesis
Mentor Graphics – CatapultC

Automatic Streaming Interfaces Between Hierarchical Blocks

Baseband Processing

Symbol Encoding
Symbol Decoding
Modulation & Spreading
Symbol Detection & Combining
Chip-Rate Demodulation & Despreading
Channel Estimation

Network Interface

Interface Synthesis for Easy Connection to Bus

Main Processor

Optimized hardware creation using Catapult Synthesis Starting from High Speed pure ANSI C++ Algorithmic Model
Multi-clock Design

- Blocks with lower data rates run with slower clock
  - Reduction in switching power
  - Reduction in static power by decreasing block area

Technology Constraints

Architectural Constraints

Multiple clocks specified with unique parameters

Clk1 = 200 MHz
  - Decimation by 2

Clk2 = 100 MHz
  - Decimation by 4

Clk3 = 25 MHz
  - Decimation by 8

Each hierarchical block can be assigned to any clock domain
Forte Design Systems – Cynthesizer

What Is Cynthesizer?

- **Standards-based SystemC synthesis**
  - Untimed C++ algorithms
  - SystemC hardware constructs
  - Algorithm and control-based designs
- ** Produces optimized RTL**
  - Doesn’t break your existing RTL sign-off flow
- **Easy to deploy**
  - Integrated process automation
  - Verification considered throughout the process
**Forte Design Systems – Cynthesizer**

**Only SystemC Synthesis Combines A High Level Of Abstraction With Required Hardware Constructs**

<table>
<thead>
<tr>
<th>Feature</th>
<th>C++</th>
<th>ANSI C</th>
<th>System Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-oriented features for managing complexity</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Custom interfaces – synthesis and simulation</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Bit-exact data types</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Fixed-point data types</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Explicit concurrency</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Synchronous logic and asynchronous logic</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Structural hierarchy</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Concurrency and protocol for control designs</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Compatible with algorithm languages</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Same simulation and synthesis semantics</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multiple levels of abstraction</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
# Forte Design Systems – Cynthesizer

## Modules & Threads
- SC_MODULE
- SC_CTHREAD
- SC_METHOD

## Ports
- sc_in<>
- sc_out<>  

## Data Types
- bool
- sc_int<>
- sc_uint<>
- sc_bigint<>
- sc_biguint<>
- cynw_fixed<>  

## C++ Constructs
- Integral C++ data types
- C++ operators
- Arrays – flattened or unflattened
- Control structures
  - if/else constructs
  - for() and while() loops
  - switch() constructs
  - references
  - statically determinable pointers

## C++ Constructs
- Structures
- Classes
- Inheritance
- Operator overloading
- Templates
  - template classes
  - template functions

Compatible with OSCI Synthesis Working Group draft synthesis subset
Production Design Success

**Application Domains**

- First tape-out 2003
- In broad use on high-volume production chips
- Chips up to 10M gates
  - 40% done with Cynthesizer
- Datapath designs
  - Video, audio, wireless, etc
- Control-dominated designs
  - Special-purpose processors
  - Memory controllers
  - I/O controllers (eg USB, SATA)

"Forte’s Cynthesizer has demonstrated its ability to quickly synthesize high-quality RTL from complex algorithms without sacrificing quality of results."

Fumiaki Nagao, Sanyo Electric
Cadence – C-to-Silicon Compiler

System-Level Model
C/C++, SystemC

C-to-Silicon Compiler
Embedded Encounter RTL Compiler

Scripts
CALYPTO
SLEC

Early Software Development

Design Constraints
Technology Library

FHM
RTL

Incisive Verification
Encounter Digital Implementation

Side-by-Side Verification

Input Model
RTL
SystemC Wrapper

Testbench

Copyright: Cadence Design Systems (www.cadence.com)

WILHELM-SCHICKARD-INSTITUT Computer Engineering

UNIVERSITÄT TÜBINGEN
NEC – Cyber Workbench

- SoC multi modules design (All-in-C)
- Input: SystemC, ANSI C (BDL), Legacy RTL
- Output: Cycle-accurate model (C++, SystemC, SpecC, Verilog), synthesizable RTL (VHDL, Verilog)
- Stimulus for behavioral simulation can be used for cycle or RTL simulation
- High controllability by constraint editor
- Automatic architecture explorer
  - Loop unrolling, pipelining
- Cycle accurate simulation with original untimed C code
- C-RTL equivalence prover (static & dynamic)
- Integrated model checking by comparing the output sequences gen. by the same stimuli
- RTL FloorPlanner
Outline

- EDA Challenges
- System Level Design
- Verification
- Synthesis
  - HLS Examples
  - HLS and Dynamic Reconfigurability
- Conclusions
Algorithm:

**Input:**
Vector with 256 8 bit values

**Output:**
Vector with 256 values and estimation coefficients

```
mean
0 - VEC_LENGTH

std
0 - VEC_LENGTH

norm
0 - VEC_LENGTH

for/for
0 - N & i - VEC_LENGTH

for
0 - N

for/for
0 - N & 0 - N

for/for
0 - N & 0 - VEC_LENGTH
```
Synthesis Goals

- 64 data vectors
- Latency: 20 ms
- Hardware: Xilinx Virtex FPGA
- Clock frequency: 75 MHz
- External memory
Manual Float/Fix Transformation

- `ac_fixed<width, integer, sign, quant, overflow>`
- `ac_int<width, sign>`
- `#include<ac_fixed.h>`
- Together with native C/C++ data types
- Word length optimized by simulation
- Simulation speed drops and memory increases
- In our example:
  - float-Version: 1377K
  - ac_fixed-Version: 19M
Optimization: 64 vectors in < 20 ms
Optimization: 64 vectors in < 20 ms
Catapult: High Level Synthesis Results

Goal: 20ms
- SCVerify option
- Starts from initial C++ specification
- Single source for simulation and synthesis
- GCC integrated
Outline

- EDA Challenges
- System Level Design
- Verification
- Synthesis
  - HLS Examples
  - HLS and Dynamic Reconfigurability
- Conclusions
From Static to Dynamic Reconfigurability

(from statically) reconfigurable

prototyping

execution

reconfiguration

minutes to days

small volume productions

execution

product lifetime

WILHELM-SCHICKARD-INSTITUT

Computer Engineering
From Static to Dynamic Reconfigurability

- (statically) reconfigurable
- dynamically reconfigurable
- multi-context
- processor-like reconfigurable
  (reconfiguration in each clock cycle)

- optimize area and performance
- quantify the benefits and costs

→ Example: NEC STP (DRP)
Evaluation Approach

- application
- performance constraint
  - initiation interval (II)

application mapping

- no processor-like reconfiguration
- utilize processor-like reconfiguration

mapping techniques for data flow
mapping techniques for control flow

compare required area and delay
Techniques for Data Flow

multi-context pipelining

<table>
<thead>
<tr>
<th>constraint</th>
<th># contexts</th>
<th># PEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>II=3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

clock cycle $n$
- state 1
- context 1

clock cycle $n+1$
- state 2
- context 2

clock cycle $n+2$
- state 3
- context 3
fast fourier transform (FFT)
(Cooley-Tukey algorithm)

Optimization of Performance
Optimization of Performance

fast fourier transform (FFT)
(Cooley-Tukey algorithm)

II=4
# of contexts 4
Optimization of Performance

fast fourier transform (FFT)
(Cooley-Tukey algorithm)

constraint
\begin{align*}
\| & = 4 \\
\text{# of contexts} & = 4 
\end{align*}

\begin{align*}
& 17 & 26 & 9 & 27 \\
& 2 & 19 \\
& 4 & 5 & 3 & 14 \\
& 3 & 24 & 3 & 35 \\
& 3 & 4 & 3 & 29 
\end{align*}
Evaluation of Performance using STP (Stream TransPose from NEC (C-based reconfigurable core))

C-based programmability and H/W level performance

- PE(8bit)x256
- 32 context plane
- MULx32
- 2port MEMx56
- 1port MEMx16
Industrial application example of STP Technology

- AV/IT Media Core Processor for Professional Camcoder
  - STP Engine is embedded as a generic accelerator of the CPU
  - Implemented functions:
    - Stream Packet Mux/DeMux
    - Audio Encode
    - Intelligent DMA
    - Video Codec, etc

Extended to the enhanced models by updating functions running on STP Engine
NEC – Cyber Workbench

Behavioral Synthesizer

- Control-intensive
- Data-dominant
- Controlflow-intensive

Behavioral description

- SystemC/SpecC
- ANSI-C (BDL)

Behavioral IP library

CPU Bus I/F generator

RTL IP

- Verilog/VHDL

Formal Verifier

- C-RTL Equivalence Prover
- Property Checker
- Library Characterizer
- QoR Analyzer
- RT Power Estimator

High-speed simulator

- Bit-accurate Behavioral Simulator
- Cycle-accurate HW/SW Co-simulator
- FPGA Accelerator
- SystemC simulator

Software

- C
- SystemC

Logic synthesis & Back-end implementation

- FPGA
- CB-IC
- G/A
- DRP

WILHELM-SCHICKARD-INSTITUT  Computer Engineering

UNIVERSITÄT TÜBINGEN
Advantages of Dynamically Reconfigurable Processors

- significant reduction of area (for our example by **up to 63%**)  
  
- mapping techniques are applicable to a wide range of applications  
  
- redirecting communication through time domain can compensate for the reconfiguration overhead
Outline

- EDA Challenges
- System Level Design
- Verification
- Synthesis
- Conclusions
Automatic Verification and Synthesis has to Become Reality!

- Design automation to increase design productivity
  - Most of design effort is in employee cost (about 80%)
  - Efforts doubles after two process generations
- Include embedded software
  - Software will get more important → 50% is software cost
  - Future systems: COTS-IP + synthesized hardware and software to keep flexibility, increase productivity and reduce risks
- Closer links with application
  - Executable specifications
  - Link requirements with verification
  - Automatic synthesis and verification for hardware and software
- How will fabless/fablite companies differentiate?
  - Application knowledge
  - EDA flow and tools