

# IDesignSpec

*Automatic generation of OVM/UVM registers*

# Agenda

## Automatic Testing

Word

Registers

OVM

UVM

SystemRDL

VMM

IDesignSpec

Frame

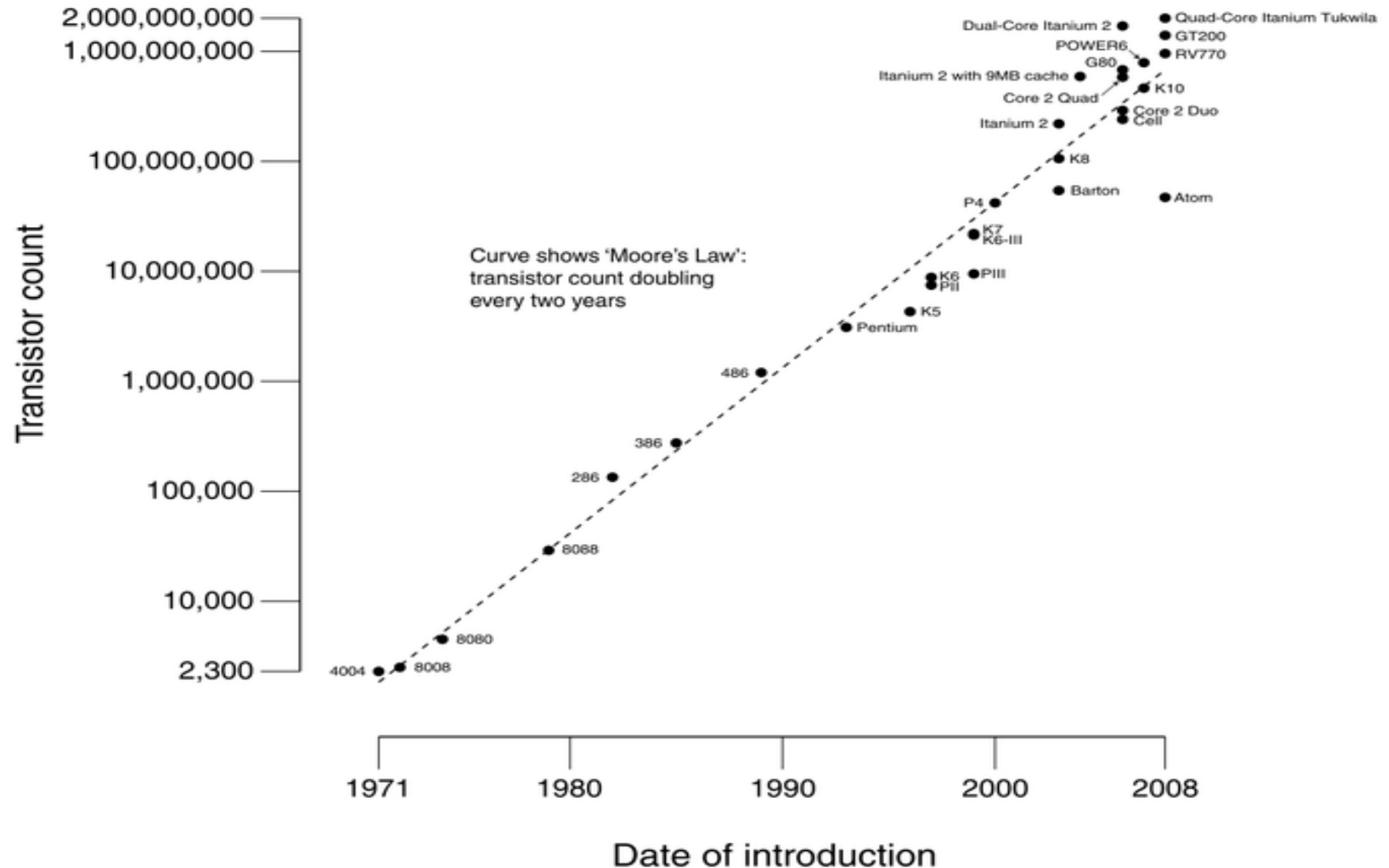
IP-XACT

RALF

Legacy Scripts

XML

# CPU Transistor Counts 1971-2008 & Moore's Law



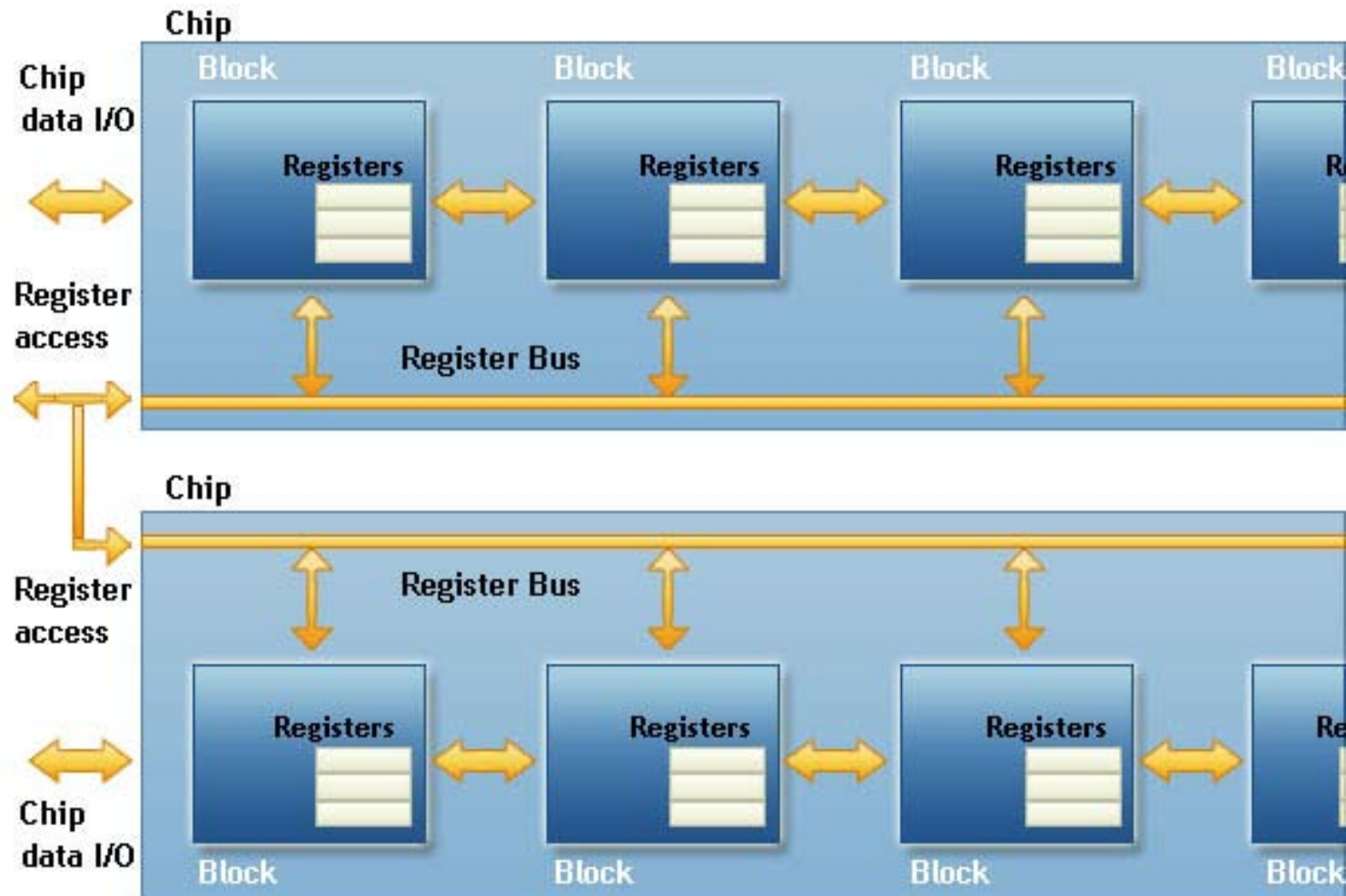
# Trends in the industry

- Higher density
  - Integration of higher density IP in an SoC
- Greater reuse of IP
  - External, 3<sup>rd</sup> party
  - Internal, older projects, other teams
- Greater configurability
  - Use of programmable registers

# Use of Programmable registers

- Control Hardware
- Send Status to Software
- Configuration tables
- 1000s of registers per IP

# Configurable Registers



# Register Fields

**Register:** Block\_b1\_config  
**Address:** 0xA0005001  
**Purpose:** Configuration of the B1 block.  
**Fields :** F1, F2



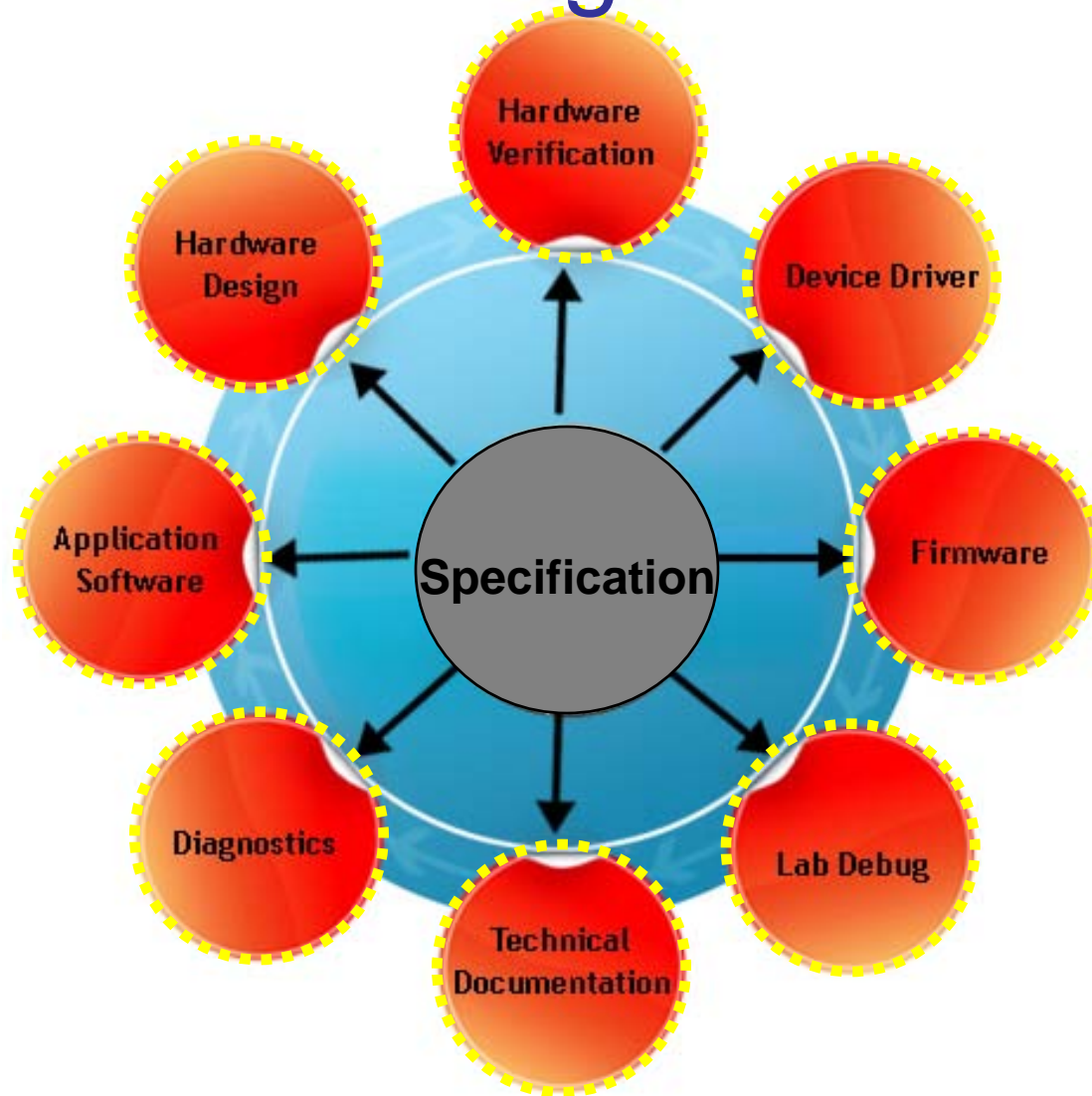
# Why are Registers Important

- Registers are present in large numbers in a chip
  - For instance VGA/LCD controller contains 560 registers
  - Microprocessor like UltraSparc contains thousands of registers
- They present programming interface
- Verification of these registers are important as their malfunction can cause failure in the whole system.
- Needs some verification methodology to address register verification challenge

# What you do with registers

- ✓ Describe
- ✓ Create in HW
- ✓ Verify
- ✓ Debug in Lab
- ✓ Program with SW
- ✓ Document for end-user
- All this takes > 20% of development time and resources

# Consumers of Register Information



# The Problem

Application	Format
Functional Specification	MS Word/Frame/OpenOffice
HW Design	VHDL/Verilog
HW Programming Guide	MS Word/Excel
Verification Environment	OVM, VMM, SystemVerilog → UVM
Firmware	C/C++ header
Diagnostics	C/C++ header
Application Software	C/C++ header



# Semiconductor Industry Challenges

- *80% of designs go over deadline by an average 44%\**
  - **70% effort devoted to verification**
  - *80% of product cost is head count related\**
- 
- Make people more effective
  - Make processes efficient
  - Get appropriate tools

\*Source: NXP/Spirit 2008

DAC10 Booth #359

Agnisys

[www.agnisys.com](http://www.agnisys.com)

# How to improve TTM & reduce costs?

- Make people more effective
  - Focus on core competency
  - Automate mundane tasks
- Make process efficient
  - Do not duplicate effort, use single source – eliminate bugs and need to keep things in-sync
  - Manage change
  - Take a holistic approach : not strive for local minima, achieve company-wide minima
- Get appropriate tools
  - Correct-by-construction : Cheapest way to catch bugs
  - Very short learning curve : low ramp-up time

# IDesignSpec

- Register Automation and Management tool
  - Simple plug-in that xforms Editor into EDA tool
    - Not based on a separate GUI or new language
  - Enables you to embedded Register information in an executable form inside Hardware Spec.
  - Generates all views from the single source
- Recipient of “Innovative EDA tool” award at VLSI Conference Jan’09

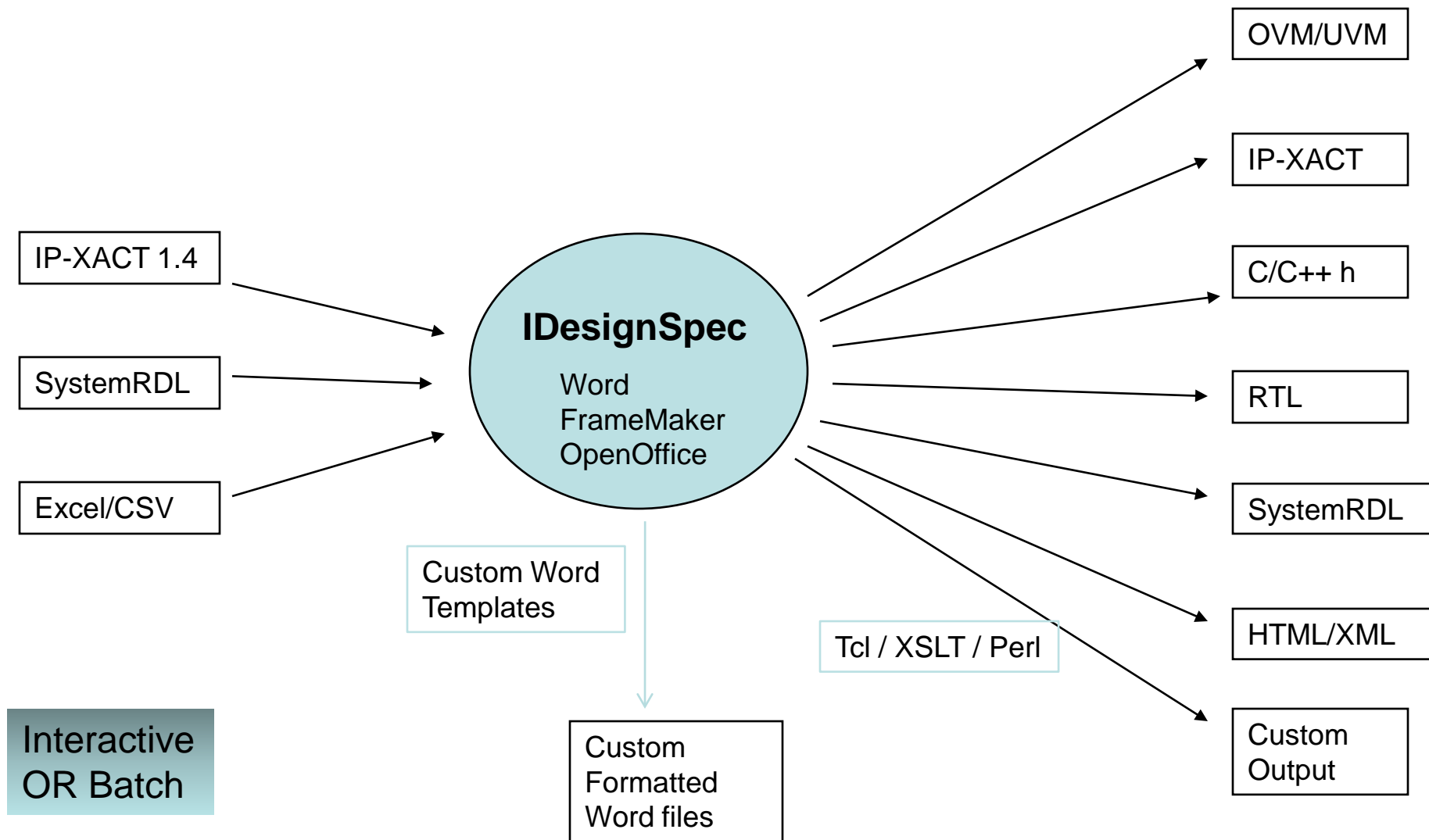


# Key Benefits

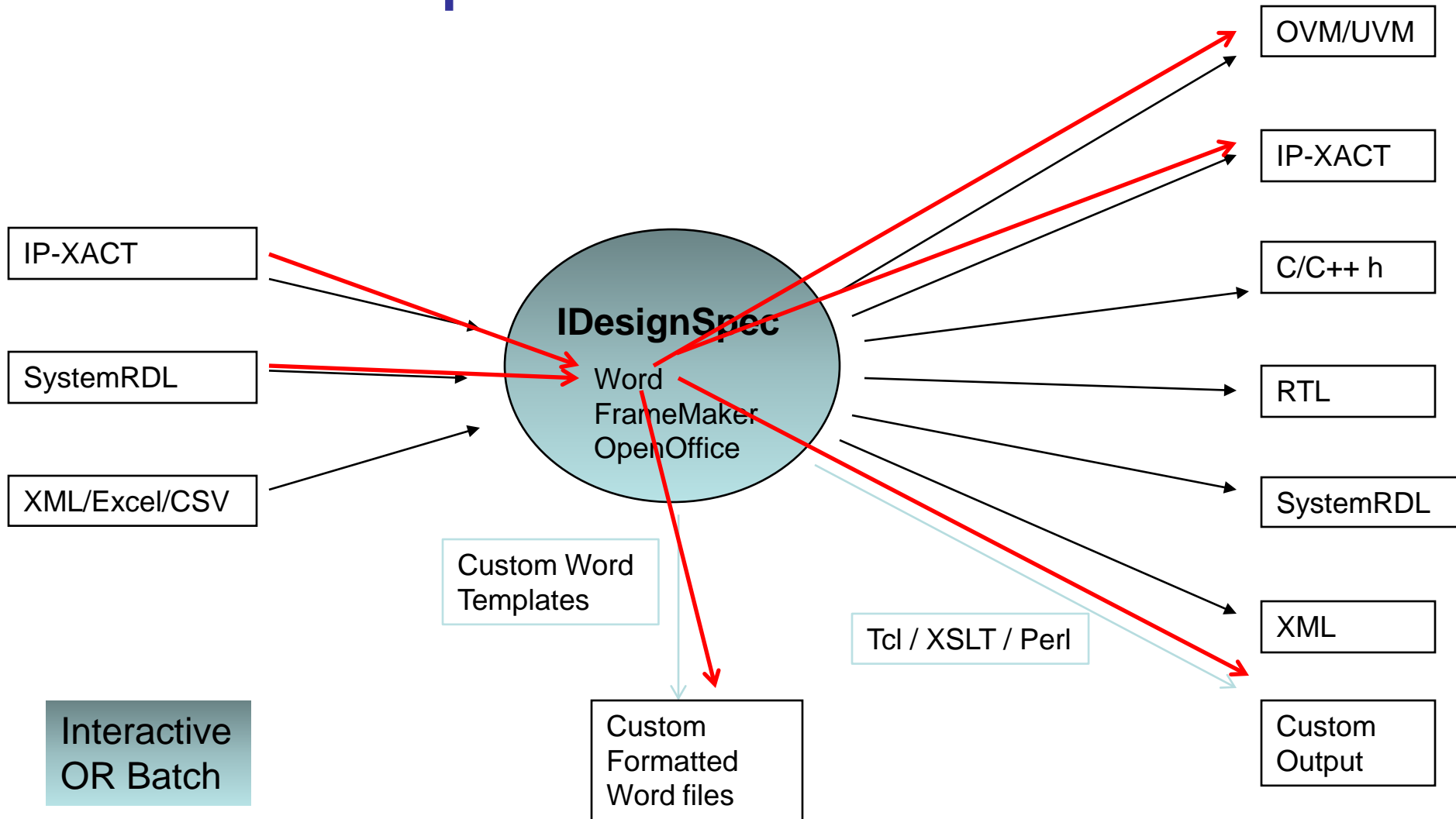
- Automatically Verify all registers in the design
- Create synthesizable code for all registers
- Get a jump start for Device Driver, Firmware and application software development
- Automatically create documentation for customers and Tech-Pubs
- Improves productivity of engineers and quality of results
- Prevents errors from entering the system
- Uses open industry standards

Data is not locked into some proprietary vendor standard

# Architecture



# Popular Use Model



# Outputs

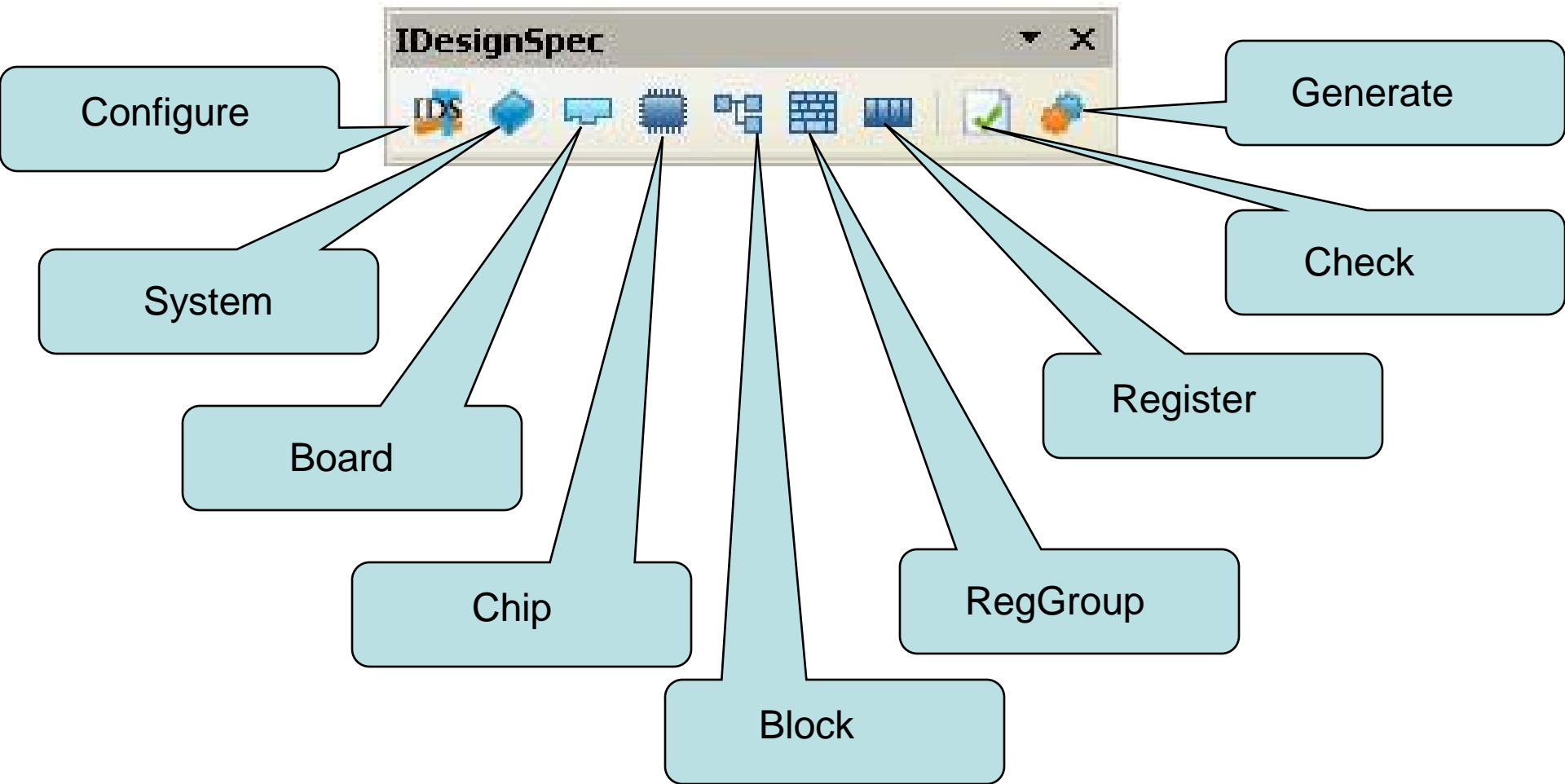
- Verification : OVM, VMM, UVM\*
- Design : SystemVerilog, Verilog, VHDL  
(AMBA®, Avalon®, any proprietary bus)
- Software : C/C++ header files
- Documentation : HTML, MS Word®, OpenOffice.org,  
FrameMaker®
- Standards: IP-XACT®, SystemRDL®

\* When it is released

# Key Advantages

- Editor based, no GUI, no new language
  - short learning curve Consistent
- Register Definition
  - Device documentation
- Enables reuse of specification and code
  - Development groups
  - Product lines
- Changes propagate automatically
  - No manual, laborious work
  - High quality
- Import existing register documentation into IDS system

# IDesignSpec: Digging Deeper



# IDesignSpec: Register



Sets specific address

1.1 sample_reg		sample_reg		Reg.	0x000004
offset	1	external			
bits	name	sw access	hw access	default	description
31:4	field1	ro	rw	0xFFFFFFFF	Field1 description.
2:1	field2	wo	ro	0	Field2 description. Enum values can go in here.

Dynamic Address Update

Dynamic update of Bit allocation table

Add fields for reg

# IDesignSpec: Document

**Special-Purpose Registers**

The special-purpose registers of all units are grouped into thirty-two groups. Each group can have different register address decoding depending on the maximum theoretical number of registers in that particular group. A group can contain registers from several different units or processes. The SR[SM] bit is also used in register address decoding, as some registers are accessible only in supervisor mode. The l.mtspr and l.mfspr instructions are used for reading and writing registers. An OpenRISC 1000 processor implementation is required to implement at least the special purpose registers from group 0. All other groups are optional, and registers from these groups are implemented only if the implementation has the corresponding unit. Which units are actually implemented may be determined by reading the UPR register from group 0.

A 16-bit SPR address is made of 5-bit group index (bits 15-11) and 11-bit register index (bits 10-0).

1.1 <i>block_grp0</i>	block_grp0	Block	0x0000000
offset			

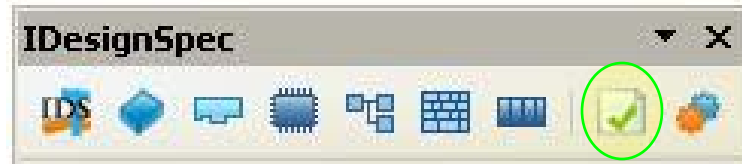
This is a set of System Control and Status registers.

1.1.1 VR	VR	Reg.	0x0000000																												
offset	external																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bits		name		sw access		hw access		default		description																					
31:0		VR		ro		rw		0		Version register																					

The SR[SM] bit is also used in register address decoding, as some registers are accessible only in supervisor mode.

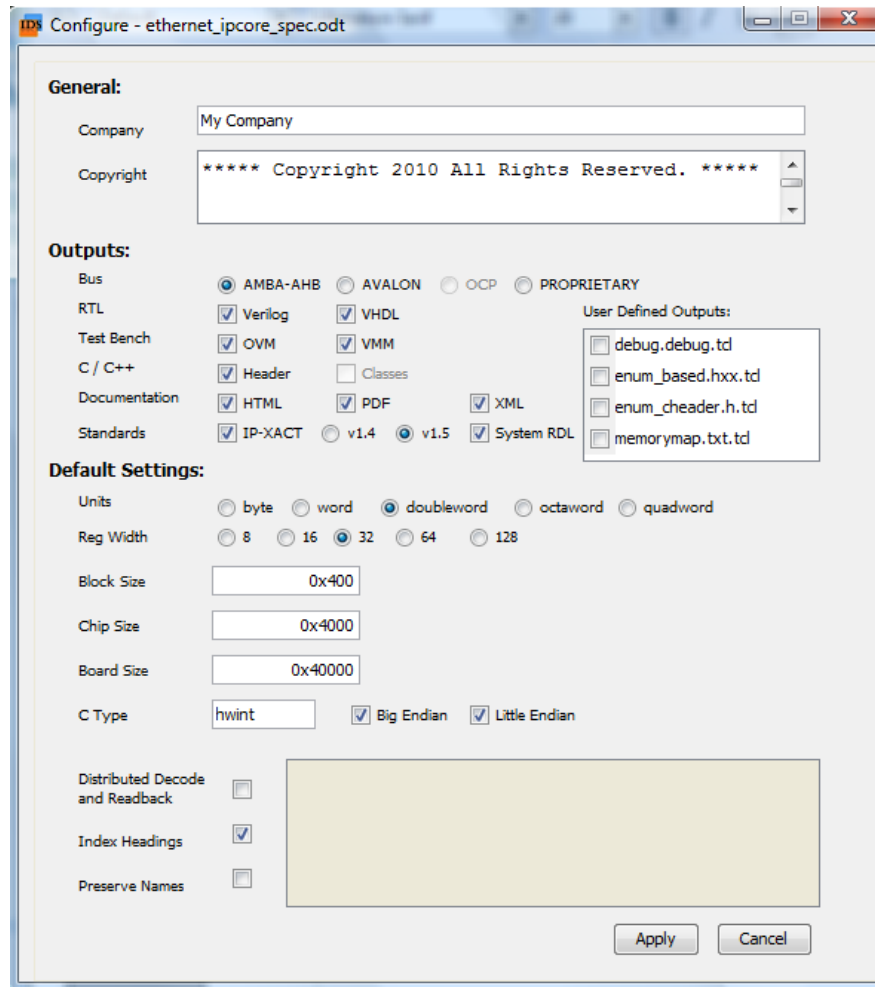
1.1.4 UPR	UPR	Reg.	0x000000C

# IDesignSpec : Check



- Consistency checks
- Address Calculation
- Error Checking (built in and custom)
  - Detects overlaps
    - Bit
    - Register
    - RegGroup
    - Block
  - Incomplete data
  - Bad/incorrect data

# IDesignSpec: Configure



Select Outputs to generate

# OVM Register Package 2.0

- Creates RegDef file
  - Packed Structures for each registers
  - Classes for each register
  - Register File for each Block
  - Register Map for each Chip

# OVM Output contd.

- OVM Register Package Support
  - Modal Register
  - Fifo Register
  - ID Register
  - Coherent Register

# Support for UVM

- Will be available when UVM register package is released
- Currently OVM, IP-XACT, RALF outputs are possible for each of the big three vendors

# IDesignSpec Summary

- A simple plugin for your existing editor
- IDesignSpec abstracts out register definition
- Use outputs for your current setup
- Evolve with the industry, without reinvesting in changing infrastructure

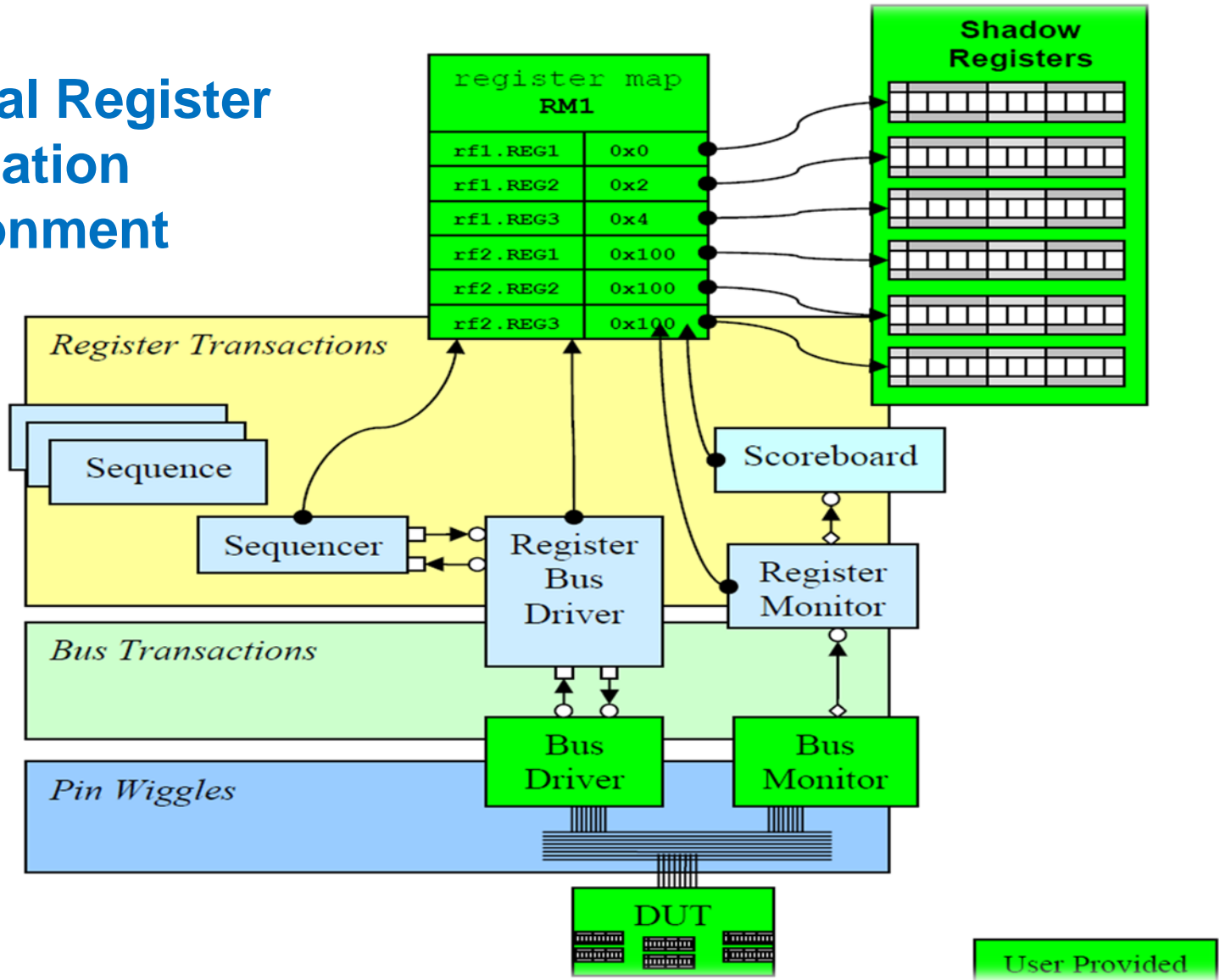
# DEMO

## Questa and IDesignSpec

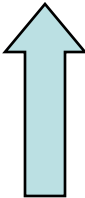
# Automated Register Verification

- **Not a new concept - been around for years.**
  - e → vr\_ad
  - Denali Blueprint (from Cisco)
  - VMM RAL
  - many others... lots of homegrown solutions
- **OVM Register is a base class library that you can use**
  - You don't have to write it yourself – like C/C++ libraries.
  - Raise abstraction level
    - Use register names instead of addresses ...
      - `write("dut.t1.register1", data)`

# General Register Verification Environment



# Register Verification Concepts

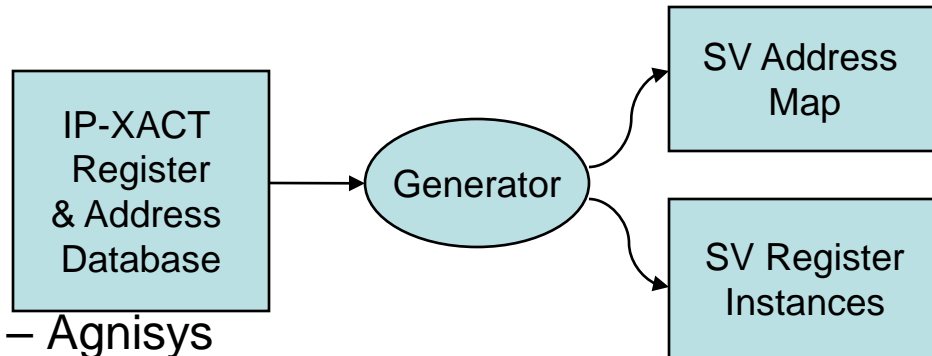
- **Do verification at a higher level**
    - **Write tests that are independent of address maps.**
      - Collect coverage, scoreboard, model checking.
    - **Raise the abstraction level of the testbench**
      - `write(REGA, 42)`
      - `write(4002, 42)`
      - Pin wiggles
-  Raising abstraction
- **Reuse register tests, coverage etc.**

# Register Generation

- “I’ve got tons of registers”
  - Don’t write them all by hand → Use a generator!
- “I don’t have XML, but I do have RTF, XLS, etc.”
  - Custom generator – most people already have one. Modify.

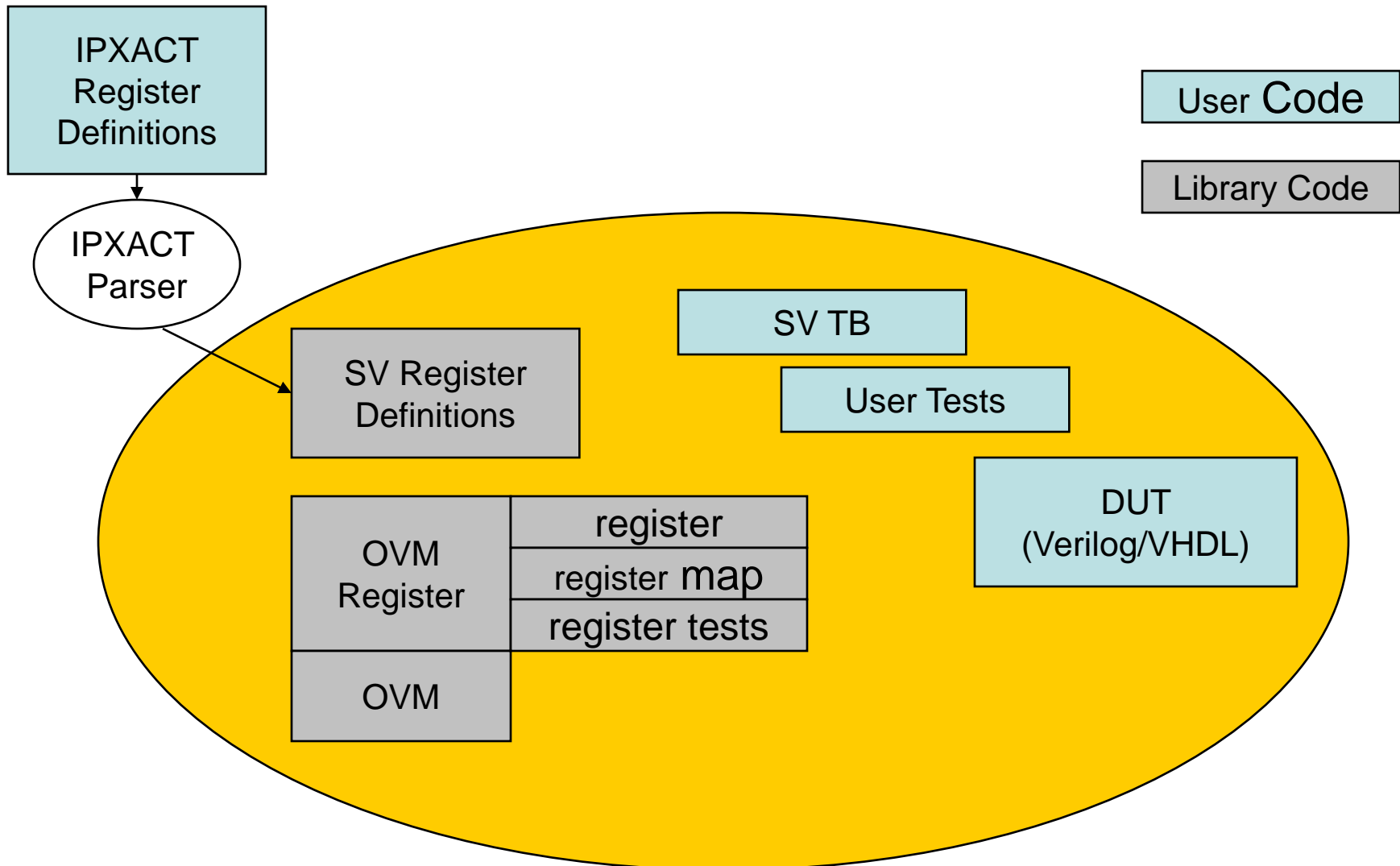
- SystemRDL
- IPXACT
- Excel Spreadsheet
- Custom

– Third Party Solutions – Agnisys



- OVM Register will contain an open source IPXACT register definition generator in the future.

# Typical User Environment



# OVM Register Definition

- **Describe each registers types**
  - Define register data fields.
  - Define field access.
  - Define field relationships.
- **Define register objects**
  - Specialize the parameterized (templated) register base class with a bit vector or structure representing the bit fields.
  - Define functional coverage for the register
  - Define random constraints for the register.
- **Define register file**
  - Map the registers to specific addresses, usually in a “register file”.
  - Map register files together into a larger “register map” describing one or more address spaces.
  - Define random constraints for the relationships between registers.

# A simple test - Simple iteration

```
class simple_test extends ...;
  register_map rm = new("rm");
  ovm_register_base r;
  BV data, rdata;
  BV addr;
  register_array_t registers = rm.get_register_array();

  // Issue hardware reset.....

  // Run test.
  foreach (registers[i]) begin
    r = registers[i];
    addr = rm.lookup_register_address_by_name(r.get_full_name());

    r.reset(); //Reset the shadow
    send_to_bus("READ", addr, rdata); // Read from the hardware
    r.bus_read(rdata); // Check the results

    r.write(`haaaaaaaaa); // Update the shadow...
    data = r.peek(); // Grab the shadow value
    send_to_bus("WRITE", addr, data); // Write the value to hardware

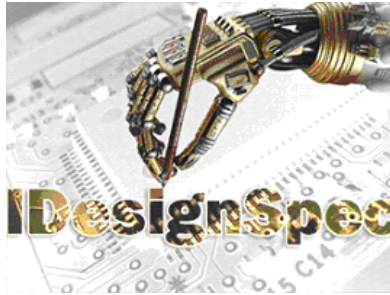
    send_to_bus("READ", addr, rdata); // Read the value from hardware
    r.bus_read(rdata); // Check the results.
  end
endclass
```

# OVM Register → Summary

- Register Base Class Library
  - Lightweight API – Open Source
  - Flexible, Easy to customize for new functionality
  - Non-intrusive, SIMPLE
- Download from [www.ovmworld.org](http://www.ovmworld.org)

# About Agnisys

- Privately held, offices in US and India.
- Founded in 2007
- What we do
  - EDA tool development
    - IDesignSpec
    - IVerifySpec
  - Verification of ASICs/FPGAs
    - Video, Audio, Image processing, Codecs, Networking, SONET



# IDesignSpec

*The Fastest Path to Code from Register Spec.*

- Try IDesignSpec in the Cloud <http://xuropa.com/agnisys>
- Sign up for a free evaluation at your site

★ **For More details : Visit Agnisys Booth #359**

Some Slides were graciously copied from Rich Edleman's (@Mentor) description of OVM Registers

# Backup slides

# OVM\_Register Class(v1.0-beta8)

- The register class that a user build registers from. This is a parameterized class which should be parameterized with a packed struct or bit vector which represents the register.

- For example

```
typedef packed struct {
```

```
    bit a[1:0];
```

```
    bit b[7:0];
```

```
} r1_t;
```

```
class r1 extends ovm_register # (r1_t);
```

```
.....
```

```
endclass
```

# OVM\_Register Class contd.

## ➤ Bit masks for permission.

RMASK: This bit mask defines bits that are readable.

WMASK: This bit mask defines bits that are writable.

## ➤ OVM Register semaphore

put() : Unlock the lock, by “putting back” n keys.

get() : Lock the lock, by “getting” n keys.

try\_get() : Like get() above, but if the lock would fail (block),  
then a zero returned instead of actually blocking.

## Field By Name Access

read\_field\_by\_name: User callable function to return the value of a field.

write\_field\_by\_name: User callable function to set the value of a field.

reset\_field\_by\_name: User callable function to reset a field by name

# OVM\_Register Class contd..

- Shadow checking routines

  - bus\_read\_bv() : This function is called when a “READ” transaction is detected.

  - bus\_write\_bv() : This function is called when a “WRITE” transaction is detected.

- Copy and clone

- Formatting/Printing

- Comparison

  - compare\_data() : Compare the data field using the compare mask.

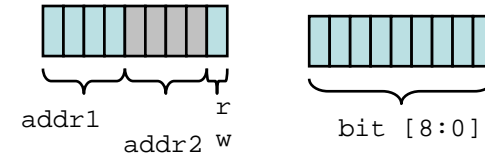
  - compare() : Same functionality as compare\_data(), but a register is passed in, instead of the register data value.

- Coverage

  - sample(): Implement this is the design specific register.

# OVM Register Data Model

- class register
  - has a value
    - the packed struct of fields  
→ the BITS.
  - read() and write()
    - “behavior” and “permission”
    - ReadOnly, ReadWrite, WriteOnly, Clear-on-Read, etc
  - registers get mapped to addresses.



```
typedef packed struct {  
    bit [3:0] addr1;  
    bit [3:0] addr2;  
    enum bit {WRITE, READ} rw;  
} my_r_t;
```

```
class my_r extends  
    ovm_register #(my_r_t);  
    ...  
endclass
```

# OVM Register Data Model / API - containers

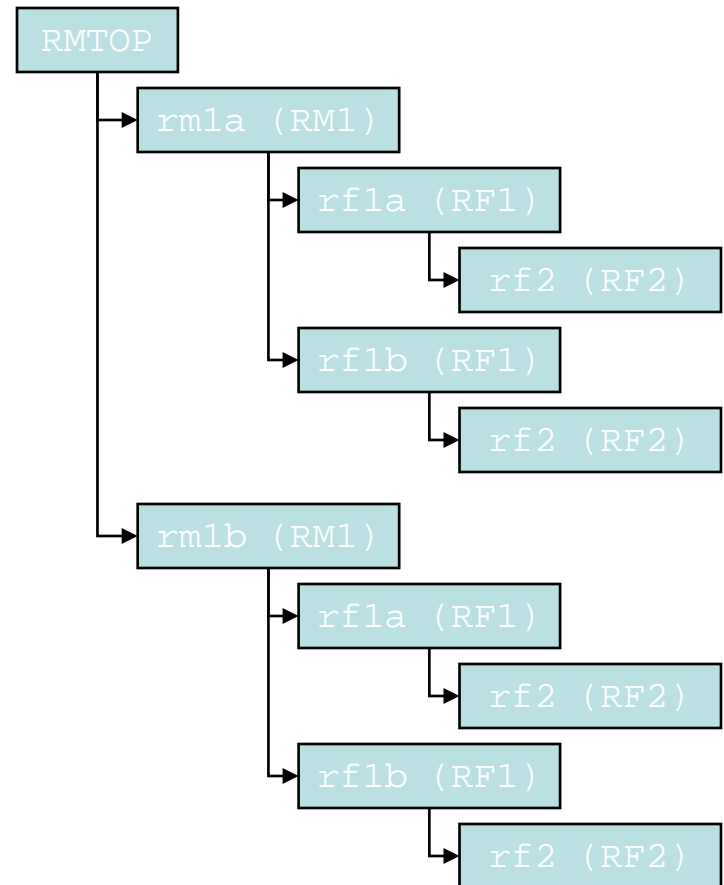
- **class ovm\_register\_file**
  - models a device or collection of registers or register files
  - “the registers on a device”
  - “contains” registers – searchable by name and address
- **class ovm\_register\_map**
  - models a system
  - “the registers in this address space”
  - “contains” register files and register maps

# Approach and philosophy

- **Simple data model with API.**

- registers are a collection of bits, most easily modeled using a SV 'packed struct'
- register files can contain registers or other register files
- register maps can contain registers, register files or other register maps

- **Use the data model and API to solve easy and hard problems**



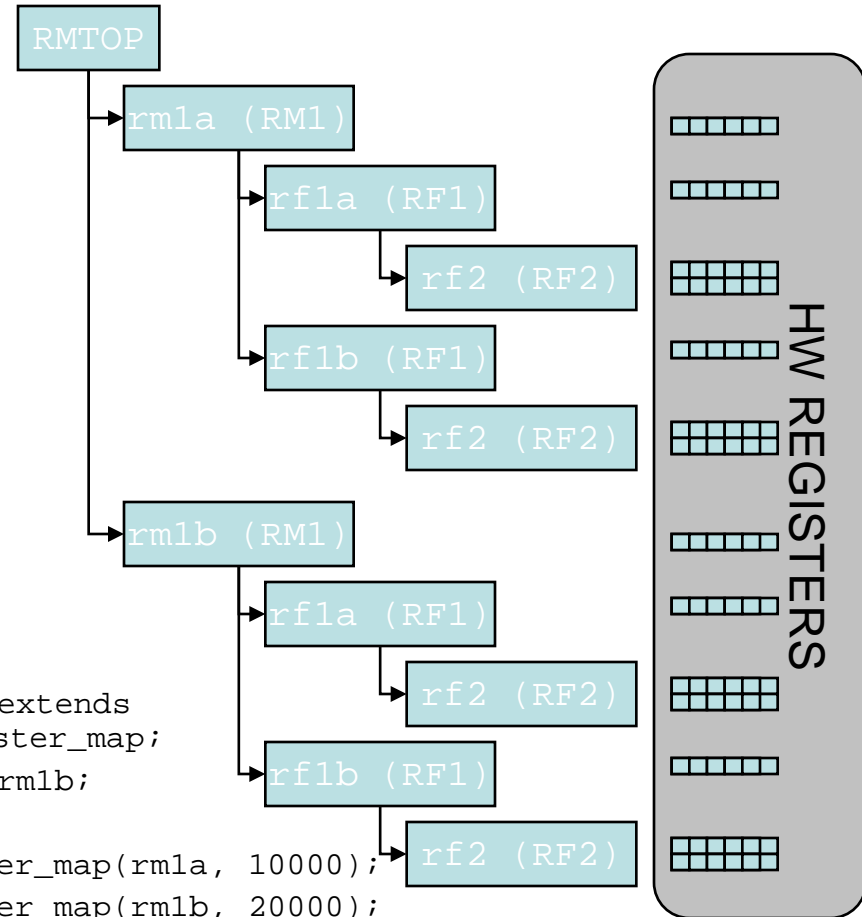
# Data model - OVM Register Definition File

```
class RF2 extends ovm_register_file;
  REGA r1, r2;
  ...
  add_register(..., 2, r1);
  add_register(..., 4, r2);
endclass
```

```
class RF1 extends ovm_register_file;
  REGA r;
  RF2 rf2;
  ...
  add_register(..., 10, r);
  add_register_file(rf2a, 100);
endclass
```

```
class RM1 extends ovm_register_map;
  RF1 rfla, rflb;
  REGA r;
  ...
  add_register(..., 0, r);
  add_register_file(rfla, 1000);
  add_register_file(rflb, 2000);
endclass
```

```
class RMTOP extends
  ovm_register_map;
  RM1 rmla, rmlb;
  ...
  add_register_map(rmla, 10000);
  add_register_map(rmlb, 20000);
endclass
```



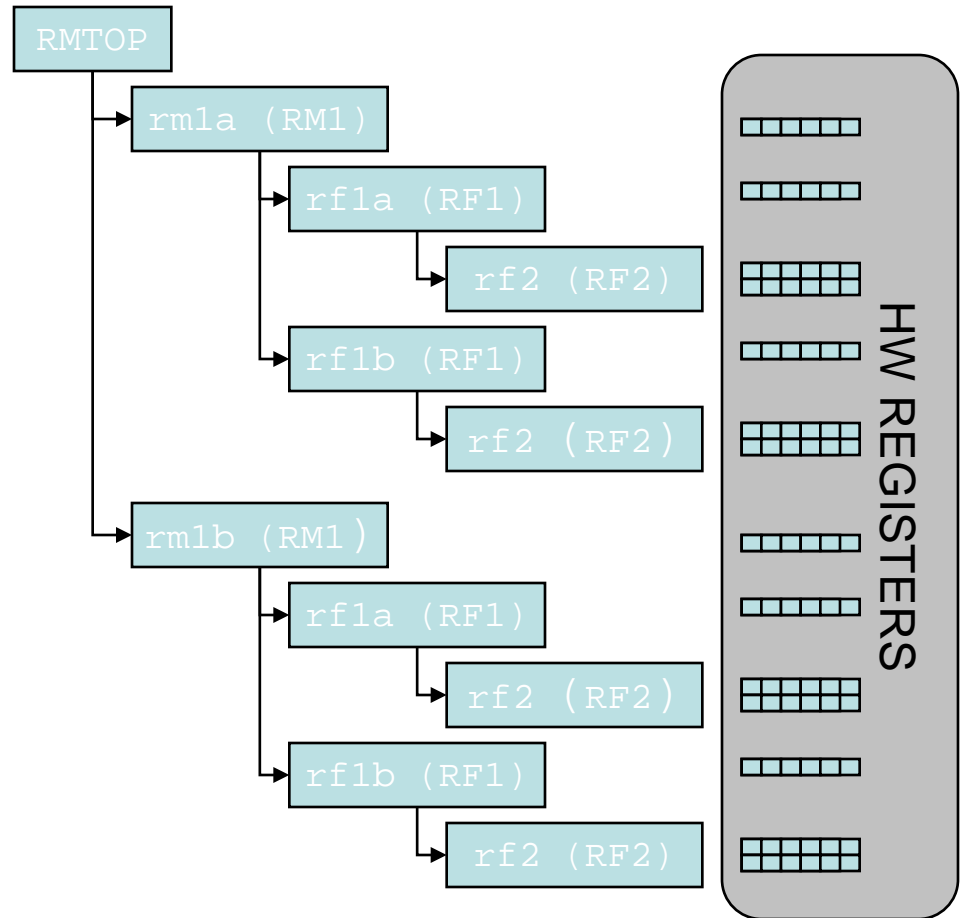
# OVM Register Definition File (2)

```
rm.rmla.r          10000

rm.rmla.rfla.rf2.r1 11102
rm.rmla.rfla.rf2.r2 11104
rm.rmla.rfla.r      11010
rm.rmla.rflb.rf2.r1 12102
rm.rmla.rflb.rf2.r2 12104
rm.rmla.rflb.r      12010

rm.rmlb.r          20000

rm.rmlb.rfla.rf2.r1 21102
rm.rmlb.rfla.rf2.r2 21104
rm.rmlb.rfla.r      21010
rm.rmlb.rflb.rf2.r1 22102
rm.rmlb.rflb.rf2.r2 22104
rm.rmlb.rflb.r      22010
```



# Defining register objects

- Registers can be bitvector.

```
typedef bit[31:0] bit32_t;
```

Register where the data is partitioned into fields.

```
typedef struct packed {  
    bit [3:0] stride;  
    bit updown;  
    bit upper_limit_reached;  
    bit lower_limit_reached;  
} stopwatch_csr_t;
```

```
class stopwatch_csr extends ovm_register  
#(stopwatch_csr_t);
```

```
...
```

```
covergroup c;  
    stride: coverpoint data.stride;  
    updown: coverpoint data.updown;  
    upper_limit_reached: coverpoint  
        data.upper_limit_reached;  
    lower_limit_reached: coverpoint  
        data.lower_limit_reached;  
endgroup
```

```
function new(...);
```

```
    c = new();
```

```
    ...
```

```
endfunction
```

```
function void sample();
```

```
    c.sample();
```

```
endfunction
```