DVCon 2012
Design & Verification Conference & Exhibition

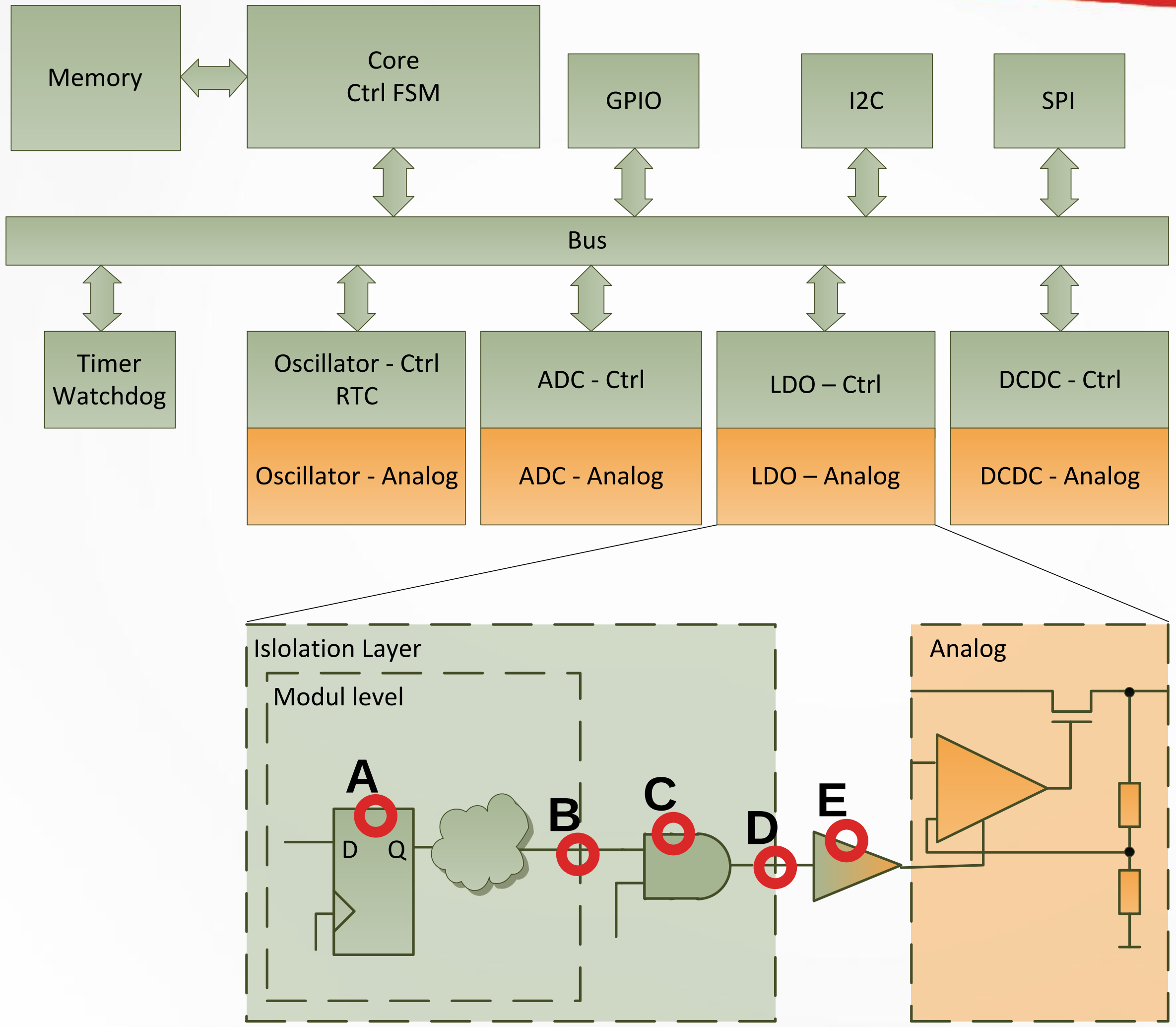
dialog SEMICONDUCTOR

Holistic Automated Code Generation: No Headache with Last-Minute Changes

Klaus Strohmayer, Norbert Pramstaller

February 28 – March 1, 2012

Motivation


Memory
Core
Ctrl FSM
GPIO
I2C
SPI
Bus
Timer
Watchdog
Oscillator - Ctrl
RTC
ADC - Ctrl
LDO – Ctrl
DCDC - Ctrl
Oscillator - Analog
ADC - Analog
LDO – Analog
DCDC - Analog
Islolation Layer
Modul level
Analog
A
B
C
D
E
D
Q

What is it all about?

Journey of a single bit

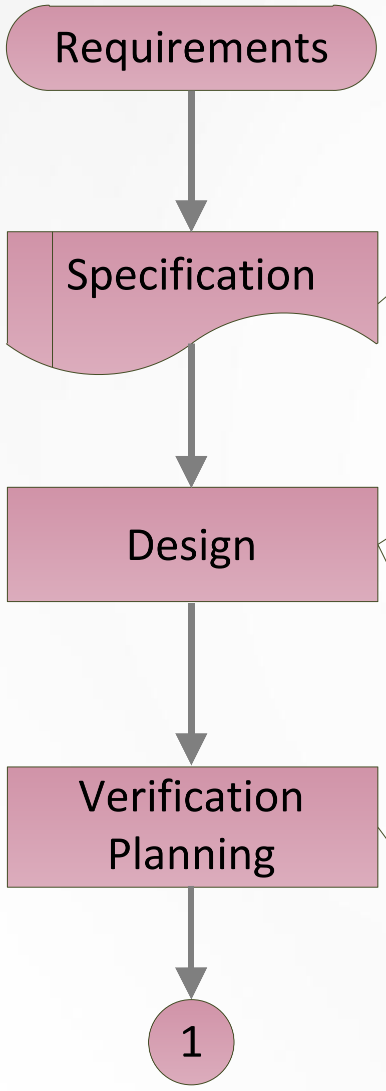
↕

Automated Code Generation

↓

Holistic Automated Code Generation

Language constructs 1/2



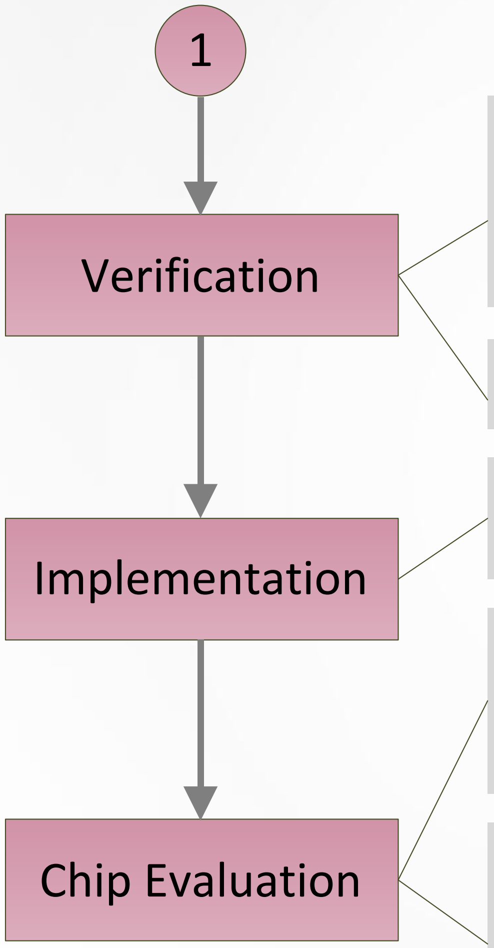
```
<reg:field id="CFG">
  <doc:desc>
    Config register
  </doc:desc>
  <reg:type read="1" write="1"/>
</reg:field>
```

```
if (wr=1'b1) begin
  case (addr)
    `CFG_ADDR: cgf <= data[5:2];
endcase
end
```

```
reg_block: SigCon_i0 (
  .cgf_q(cfg_q),
  ...
);
```

```
coverage "CFG" {
  items_pattern:
<text>(HDL,instance)SigCon_i0.cfg_reg_we
</text>;
  item_details: "Values for write
access.";
};
```

Language constructs 2/2



```
covergroup cfg_reg_we @(posedge clk);
  coverpoint data[5:2] iff (wr=1'b1);
  ...
endgroup
```

```
write_bitfield(CFG, 3);
```

```
set_load 50 SigCon_cfg_q
```

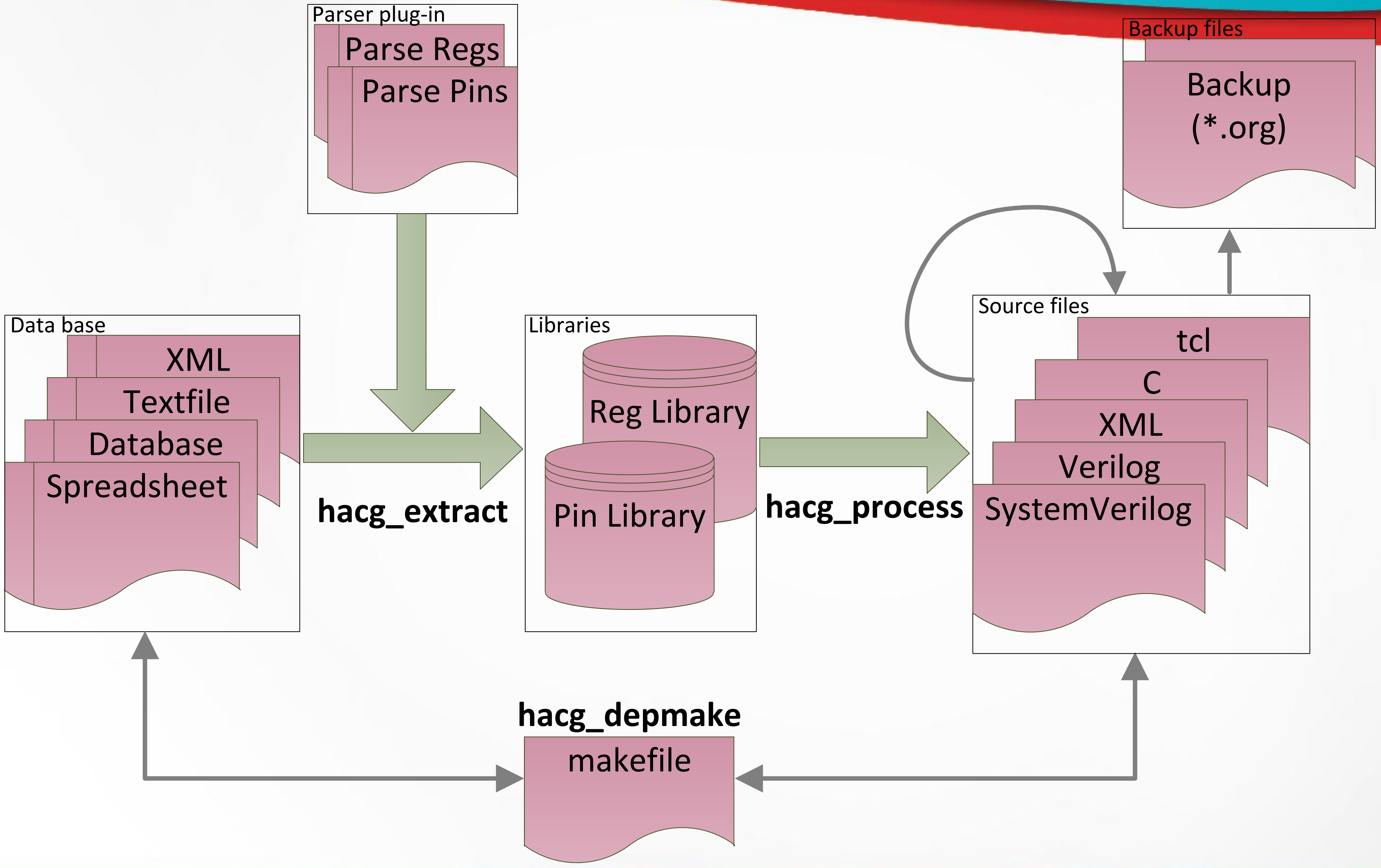
```
sfr at 0x01B1  CFG; // CFG register
  #define CFG_BF1_SHFT        2
  #define CFG_BF1_SHFT_SIZE   4
```

```
  [CFG]
  PARENT=SigCon
  NAME=CFG
  MODE=BYTE
  ADDRESS=0x01B1
  NOTE=Config register
```

HACG Prerequisits

1) Unique database
2) Automated generation of regular structures
3) Combination of handwritten and automated generated code
4) Automated dependency handling

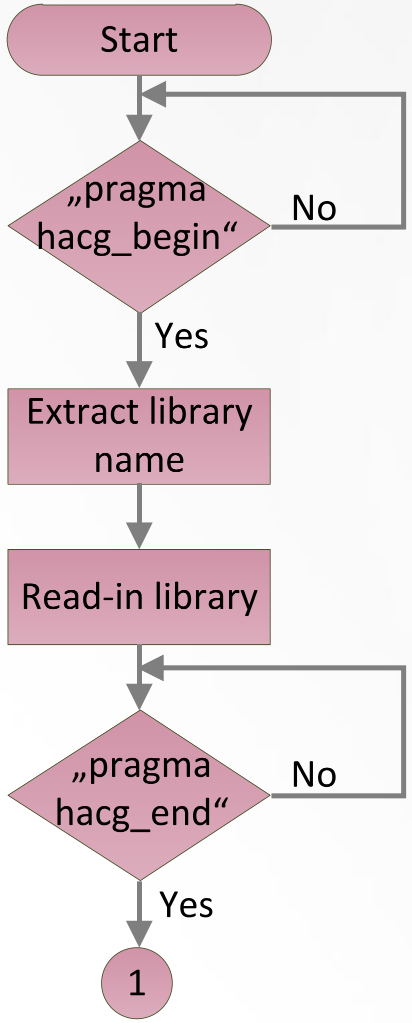
HACG Flow


Parser plug-in
Parse Regs
Parse Pins
Backup files
Backup
(*.org)
Data base
XML
Textfile
Database
Spreadsheet
hacg_extract
Libraries
Reg Library
Pin Library
hacg_process
Source files
tcl
C
XML
Verilog
SystemVerilog
hacg_depmake
makefile

HACG Tools

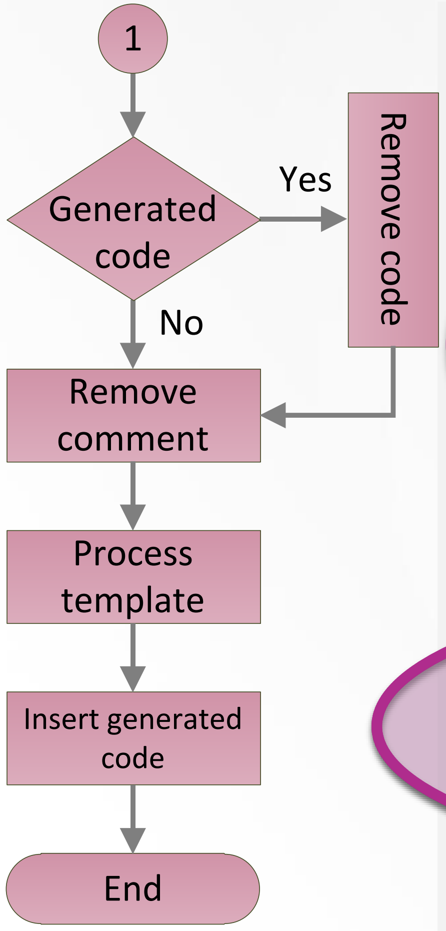
- hacg_extract
  - Extracts data based on plug-in
  - Stores data in a generic data format
- hacg_process
  - Processes source files
- hacg_depmake
  - Check dependcies of source files from data sources

HACG Process 1/2



```
always_comb begin
   if (rd == 1'b1) begin
   // pragma hacg_begin RegList.REG
   // [%- FOREACH g = grp('Name'), IF (g.Name == 'SigCon');
   //    FOREACH sf = g.reg;
   //      "\n      if ("; m.lower("reg.addr_${sf.ID}) begin");
   //      FOREACH b = sf.bit; IF (b.ID != "ni");
   //        m.format("\n        datao_b[%-20s] = %s;",
   //        "`${sf.ID}_${b.ID}_FLD", m.ffb("${sf.ID}_${b.ID}"));
   //      END; END;
   //      "\n      end";
   //    END;
   //  END; END -%]
   ////>> Start of hacg inline generated code. Don't change! ////
   if (sfr.addr_cfg) begin
     datao_b[`CFG_BF1_FLD ] = cfg_bf1_ff;
     datao_b[`CFG_BF2_FLD ] = cfg_bf2_ff;
   end
   ...
   //// End of hacg_inline generated code! <<////
   // pragma hacg_end
```

HACG Process 2/2



```
always_comb begin
  if (rd == 1'b1) begin
  // pragma hacg_begin RegList.REG
  // [%- FOREACH g = grp('Name'); IF (g.Name == 'SigCon');
  //    FOREACH sf = g.reg;
  //      "\n      if ("; m.lower("reg.addr_${sf.ID}) begin");
  //      FOREACH b = sf.bit; IF (b.ID != "ni");
  //        m.format("\n      datao_b[%-20s] = %s;",
  //        "`${sf.ID}_${b.ID}_FLD", m.ffb("${sf.ID}_${b.ID}"));
  //      END; END;
  //      "\n      end";
  //    END;
  //  END; END -%]
  ////>> Start of hacg inline generated code. Don't change! ////
  if (sfr.addr_cfg) begin
    datao_b[`CFG_BF1_FLD ] = cfg_bf1_ff;
    datao_b[`CFG_BF2_FLD ] = cfg_bf2_ff;
  end
  ...
  //// End of hacg inline generated code! <<////
  // pragma hacg_end
```

HACG Advantages

- Unique data base
- Applicable to all types of regular structures
- Supported by all scripting and programming languages

- Template commands are part of the source file
- Project-wide templates possible via INCLUDE
- Unified coding style

- Combining automated generated and handwritten code
- Easy step-by-step introduction
- Automated dependency checking and makefile generation

HACG Summary

```
if (wr=1'b1) begin
  case (addr)
    // pragma hacg_begin RegList.REG
    // ...
    ////>> ... ////
    `CFG_ADDR: cgf_bf1 <= data[5:2];
    //// ... <<////
    // pragma hacg_end
```

```
reg_block: SigCon_i0 (
    .clk(clk),
    // pragma hacg_begin RegList.REG
    // ...
    ////>> ... ////
    .cgf_bf1_q(cfg_bf1_q),
    //// ... <<////
    // pragma hacg_end,
```

CFG		0x3F1			Configuration register
	ni	[1:0]	2'b00	R	Not implemented
	BF1	[5:2]	4'h3	RW	Configuration bitfield 1
	ni	[7:6]	2'b00	R	Not implemented

```
  // ...
  ////>> ... ////
  coverpoint data[5:2] iff (wr=1'b1);
  //// ... <<////
  // pragma hacg_end
  ...
endgroup
```

```
////>> ... ////
sfr at 0x01B1  CFG; // CFG register
  #define CFG_BF1_SHFT        2
  #define CFG_BF1_SHFT_SIZE   4
//// ... <<////
// pragma hacg_end
...
```

HACG Summary

CFG		0x3F1			Configuration register
	ni	[1:0]	2'b00	R	Not implemented
	BF1	[5:2]	4'h3	RW	Configuration bitfield 1
	BF2	[7:6]	2'b01	RW	Configuration bitfield 2

HACG Summary

```
if (wr=1'b1) begin
  case (addr)
    // pragma hacg_begin RegList.REG
    // ...
    ////>> ... ////
    `CFG_ADDR: cgf_bf1 <= data[5:2];
    `CFG_ADDR: cgf_bf2 <= data[7:6];
    //// ... <<////
    // pragma hacg_end
  endcase
end
```

```
reg_block: SigCon_i0 (
    .clk(clk),
    // pragma hacg_begin RegList.REG
    // ...
    ////>> ... ////
    .cgf_bf1_q(cfg_bf1_q),
    .cgf_bf2_q(cfg_bf2_q),
    //// ... <<////
    // pragma hacg_end,
  ...
);
```

```
covergroup cfg_reg_we @(posedge clk);
  // pragma hacg_begin RegList.REG
  // ...
  ////>> ... ////
  coverpoint data[5:2] iff (wr=1'b1);
  coverpoint data[7:6] iff (wr=1'b1);
  //// ... <<////
  // pragma hacg_end
  ...
endgroup
```

```
// pragma hacg_begin RegList.REG
// ...
////>> ... ////
sfr at 0x01B1  CFG; // CFG register
  #define CFG_BF1_SHFT       2
  #define CFG_BF1_SHFT_SIZE  4
  #define CFG_BF2_SHFT       6
  #define CFG_BF2_SHFT_SIZE  2
//// ... <<////
// pragma hacg_end
...
```

HACG

Holistic Automated Code Generation:

No Headache with Last-Minute Changes