“How To Make A Chip”

Design Flow and Tools used for the Design of a Pipelined ADC

Thomas Liechti
EPFL IMM LSM

May 30, 2007
Overview

• Top-Down Design Flow Overview
• Used Tools
• Managing the Hierarchy
• Behavioral Modeling with Verilog-A
• Tips for Schematic and Layout Design
• Pad Ring and ESD Protection
• Post-Layout Simulation
• Using Verilog-A in Testbenches
• Dealing with small embedded Logic
• Stream-out and Design Rule Verifications
Top-Down Design Flow

1. **Toplevel Specs**
   - Explore algorithms and architectures using MATLAB

2. **Build hierarchical structure with behavioral blocks (HDL)**
   - Update/refine bhv models

3. **Schematic design of blocks (device level)**
   - Extract parasitics

4. **Draw Layout of blocks**
   - Refine Floorplan, Start Toplevel Layout
   - Finalize toplevel layout, do checks

5. **Toplevel Floorplan**
   - Stream out GDS

6. **Toplevel Simulations**
   - Simulate with selected blocks at device level
   - Extract parasitics

7. **Block simulations**
   - All blocks at device level (functional verification)

8. **Toplevel Simulations**
   - Full-chip extracted simulation

9. **Finalize toplevel layout, do checks**

10. **Draw Layout of blocks**
    - Refine Floorplan, Start Toplevel Layout

11. **Toplevel Floorplan**
    - Stream out GDS

12. **Top-Down Design Flow**
    - Full-chip extracted simulation

---

**ECF FÉDÉRALE DE LAUSANNE**
Advantages of Top-Down Design Flow

- Macromodels are “executable specs” for blocks. They are used in the design process and are more likely to be kept up to date than written documents.
- Good **testbenches**: each block can be verified as part of the whole system
- Block design can easily be **parallelized**, communication between designers is improved (common framework)
- Design **reuse**: macromodels exist for each block
- Avoids discontinuity between system level and circuit level design (common design representation)
Tools

- Cadence icfb (5.0.33 and 5.1.41)
- Spectre
- Assura (3.1.6) / Calibre (2006.4)
- MATLAB, Perl
- (Modelsim [Mentor], Design Compiler/Design Vision [Synopsys], Encounter [Cadence])
Use config views and the hierarchy editor to control which views are used for different blocks in the hierarchy.
Behavioral Models

• Used Verilog-A for behavioral models:
  – Verilog-A blocks can be simulated with spectre
  – Does not support Verilog (hence not well suited for modelling digital blocks)

• Alternative: VHDL-AMS
  – true mixed-signal (digital VHDL included)
  – not used because of design kit restrictions and simulator availability

• In Verilog-A describes behavior using
  – branch contribution statements (“controlled sources”)
  – indirect branch contribution statements (implicit equations for branch potentials and flows)
  – procedural assignment statements for variables
Verilog-A Code Examples

**Ideal 14-bit DAC:**

```verilog-a
`include "constants.vams"
`include "disciplines.vams"

`timescale 1ns / 100 fs

module DAC14_ideal (code, clk, sig);
  input [13:0] code;
  input clk;
  output sig;
  electrical [13:0] code;
  electrical clk;
  electrical sig;

  real value;
  integer i;

  analog begin
    @( cross( V(clk)-0.9, 0 ) ) begin
      value = 0;
      generate i (0,13)
        value = value/2 + ((V(code[i])>0.9)?1:0)*0.5;
      end
      V(sig) <+ transition(value,0,10p);
    end
  endmodule
```

**Ideal Opamp:**

```verilog-a
... analog begin
  V(out): V(in) == 0;
end
...
```

- **Indirect branch contribution statement**
- **Procedural assignments**
- **Branch contribution statement**
Modeling Difficulties

• One of the main obstacles to the described top-down flow: *Even simple analog blocks are difficult to model*

• Interface problem: difficult to accurately model loading effects (tight coupling between analog blocks)

• For example MDAC:
Schematic Design

- Follow naming conventions (prefixes, bit order, etc.)
- Annotate schematics (branch currents, dummy devices, critical nodes, etc)
- Use meaningful instance and signal names (e.g. MDAC, OTA, COMP)
- Don’t spend a lot of time optimizing the schematic. Complete a first version of the schematic quickly (add some parasitic capacitance to critical nodes), then draw a first version of the layout and resimulate the block with back-annotated parasitics. Modify the schematic if necessary and iterate until specs are met.
Drawing the Layout

• Build layout from parameterized cells (PCells) of devices
• Use paths for simple routing (be careful with 45° angles!)
• Multipart Paths for more complex structures (shielded lines, slotted metal lines, “path contacts”, etc.)
• Write custom PCells, e.g. for bypassing MOS caps.
• Regular structures make life easier: draw subcells and use them to build larger structures. Use “mosaics” for robust arrays
• Toplevel:
  – Keep revising the initial floorplan while block layouts designed
  – Draw pad ring and ESD network early (number of pins, available space)
Hierarchical PCell Example

- **Goal:** Fill rectangular area with mosaic of PMOS caps
- **Parameters:** bounding box, max. sub-cell dimensions and max. poly finger length.

(Number and dimensions of sub-cells, and number and length of fingers per subcell are calculated by the pcell code based on the primary parameters)

```plaintext
cDefinePCell(                                        
  ;--------------------------                        
  ; Identify target cellView                        
  ;--------------------------                        
  list( ddGetObj("MADC_Fab1B")                      
    "mosaicMOS_Cell" "layout" )                      
  ;--------------------------                        
  ; Formal parameter names                           
  ;--------------------------                        
  (                                                      
    (cellW 20.0)                                     
    (cellH 10.0)                                     
    (maxL 3.0) )                                     
  ;--------------------------                        
  ; The code drawing the geometry                    
  ;--------------------------                        
  let( (effL fingers
```

compile and instantiate
Pad Ring and ESD Protection

- No I/O library provided by foundry: build I/O ring and ESD protection from basic blocks (pads, clamps, diodes).
ESD Protection Network

- All supply domains connected to common discharge line (CDL) through cross-coupled ESD diodes.

Clamps triggered by ESD transient
Parasitic Extraction & Post-Layout Simulation

• Create av_extracted view with Assura RCX, use this view instead of schematic view for netlisting.
• R extraction fractures nets. To probe signals at specific device terminals, used a perl script to find the fractured net names in the extracted netlist.

node identifier file:
FENC.C2C(\d+).M40 1 "Flash output (p)"
FENC.C2C(\d+).M46 1 "Flash output (n)"
...

extracted netlist:
STG1\|FENC\|C2C\(\d\(5\)\)\|M25 (_57452:vss_COM \
_6:\STG1\|FENC\|C2C\(5\)\|nd STG1\|FENC\|C2C\(5\)\|nf vss_COM) \
nfet w=1.2e-06 l=1.8e-07 as=0.32401p ad=0.32401p ps=1.73998u \
pd=1.73998u nrd=1.442e-01 nrs=1.442e-01 m=1 
STG1\|FENC\|C2C\(5\)\|M24 (STG1\|FENC\|C2C\(5\)\|nf 
...

spectre save statements:
# Flash output (p) (FENC.C2C(\d+).M40:1)
save \nI246.I0._2:\STG1\|th\(5\)\ 
I246.I0._2:\STG1\|th\(3\)\ 
...

perl
Testbenches

• Use Verilog-A for “instrumentation modules”
  – measure quantities (output as node voltage or write to file for post-processing in MATLAB for example)
  – generate stimuli

• Of course, this could also be done using Analog Environment Calculator or OCEAN
Example: Settling Time Measurement

MDAC (DUT)

Verilog-A measurement block
Example: Setting Time Measurement

Internally, the Verilog-A block uses a delayed version of the input voltage to determine the final value before measuring the settling time.

MDAC settling time vs input voltage for two different output loads
Example: Bit Logger

- Sample CML logic output of ADC and write to file
- Analyze data in MATLAB
Example: Stimulus Generation

- Parameterizable state machine
Small (Synthesized) Logic Blocks

- Small amount of logic needed to control switches in MDAC
- Logic was generated from Verilog:
  - RTL level verilog code
  - Design Compiler
  - gate level verilog code
  - Encounter P & R
  - GDSII layout

- Work-around to be able to simulate whole design with spectre and perform LVS with Assura:
  - GDS II layout
  - extract transistor level netlist (Calibre)
  - calibre view
  - netlister
  - spectre simulation
  - cdI netlist
  - Assura LVS
Stream Out & Verification

- Toplevel LVS
- Translate the layout data from DFII to GDSII (Stream out).
- Run final checks on GDSII database:
  - DRC
  - Antenna
  - Stress
  - ERC
  - ESD/Latchup
- Usually a good idea to run Assura/Calibre in batch mode, i.e. prepare input files (.rsf, .rules) and run tools from command line
  - easy to repeat same run, can run multiple checks at same time
References


• Chapter 2 in *Trade-Offs in Analog Circuit Design*, Toumazou/Moschytz/Gilbert, Kluwer 2002