Physical Design via Place-and-Route: RTL to GDS

Edward Wang
April 10, 2018
• Stands for Register Transfer Level
• An abstraction for digital circuits, consisting of
  ○ Combinational logic
  ○ Registers (state elements)
  ○ Modules (hierarchical and “blackbox” - e.g. analog macros, SRAM macros, etc) and ports/nets
• Described in terms of a hardware description language (HDL)
Hardware description languages (HDLs)

- An HDL is a language for describing circuits using the RTL abstraction.
- Includes facilities for describing combinational logic, registers/state, and modules.
- Common HDLs: Verilog, VHDL.
- Research-y HDLs: FIRRTL, CoreIR.

```plaintext
module add_one :
    input clock : Clock
    input reset : UInt<1>
    output io : {flip in : UInt<4>, out : UInt<4>}

    io.out <= tail(add(io.in, UInt<1>("h01")}, 1)
```
circuit HelperDelayedAdd2 :
  module HelperDelayedAdd2 :
    input clock : Clock
    input reset : UInt<1>
    output io : {flip in : UInt<4>, out : UInt<4>}
  reg my_reg : UInt, clock
  my_reg <= tail(add(io.in, UInt<1>("h01")), 1)
  io.out <= tail(add(my_reg, UInt<1>("h01")), 1)
module HelperDelayedAdd2 (  
input clock,  
input reset,  
input [3:0] in,  
output [3:0] out);

reg [3:0] my_reg;

always @ (posedge clock) begin  
  my_reg <= in + 4'h1;
end  
assign out = my_reg + 4'h1;

endmodule
Chisel: Constructing Hardware In a Scala Embedded Language

- Hardware Construction Language (HCL)
- Started in 2010
- HCL based in Scala, a functional/object-oriented programming language
- Hosts Berkeley hardware projects (e.g. RocketChip, Hwacha, BOOM)
- Many ongoing industry collaborators (Intel, LBNL, Northrop Grumman, Google, SiFive)

```scala
import chisel3._

class GCD extends Module {
  val io = IO(new Bundle {
    val a = Input(UInt(32.W))
    val b = Input(UInt(32.W))
    val e = Input(Bool())
    val z = Output(UInt(32.W))
    val v = Output(Bool())
  })
  val x = Reg(UInt(32.W))
  val y = Reg(UInt(32.W))
  when (x > y) { x := x - % y }
  .otherwise { y := y - % x }
  when (io.e) { x := io.a; y := io.b }
  io.z := x
  io.v := y === 0.U
}
```
Digression: HDL vs HCL

- Chisel (strictly speaking) isn’t an HDL...
- What’s the difference?
Other HCLs/HDLs

- PyMTL
- Bluespec
- Magma
- Lava
- Netlist
- etc
RTL Design Is Only Part of the Picture
RTL Design Is Only Part of the Picture
What Makes Creating Hardware Difficult?

- What makes the design cycle long and expensive?
  - Architectural Design Space Exploration
  - RTL Development [J. Bachrach et al, DAC 2012]
  - Physical Design and Implementation
  - Verification - “is it correct?”
  - Validation - “is it the right problem to solve?”

- Compilers and Generators
  - Having reliable, re-usable, and robust tools is the name of the game
  - BAG, Chisel, etc.
YOU MEAN TO TELL ME

AUTOMATIC P&R ISN'T FULLY AUTOMATIC!?
Why Physical Design Matters
Physical Design is HARD - CAD Tools Aren’t Automatic

```
module Delay(
  input clk,
  ...,
  endmodule
```

**Verilog Instances**

**CAD Tools**

**Chip**

**Legend**

```
source dc_setup.tcl
set_svf ${RESULTS_DIR}/${DCRM_SVF_OUTPUT_FILE}
define_design_lib WORK -path ./WORK
analyze -format verilog ${RTI_SOURCE_FILES}
elaborate
remove_unconnected_ports -blast_buses [find -hier cell "*"]
write -hierarchy -format ddc -output ${RESULTS_DIR}/${DCRM_OUTPUT_FILE}
link
...
```

**TCL Scripts**

**source code**

**generated file**
Agile RTL is slowed by non-Agile Physical Design

```java
1. class Delay extends Module {
2.     val io = IO(Bundle(
3.         (UInt(4, W))
4.         (UInt(4, W))
5.     )
6.     )
7.     val r = Register(UInt())
8. }
```

```
module Delay(
    clk,
    [3:0] in,
    output [3:0] out
); ... endmodule
```

```tcl
1. source dc_setup.tcl
2. set_svf ${RESULTS_DIR}/${DCRM_SVF_OUTPUT_FILE}
3. define_design_lib WORK -path ./WORK
4. analyze -format verilog ${RTL_SOURCE_FILES}
5. elaborate ${DESIGN_NAME}
6. remove_unconnected_ports [find -hier cell "*"]
7. remove_unconnected_ports -blast_buses [find -hier cell "*"]
```
Digression: Why Agile Physical Design?

1. Analog/Mixed-Signal (AMS) Systems
2. Improved Usability for Faster Design Space Exploration
3. Technology Portability
4. Hierarchical Design
Research Plug
Physical design is a collection of many difficult problems
- No silver bullet

Need to lower barrier to solving these problems
- Other tapeouts solve these problems, but their solutions are not general or reusable
- Get designers to encode solutions in a more reusable way, so future tapeouts can leverage previous work (even with different technologies, CAD tools, or designs)

Provide collection of API’s that designers leverage to build these tools
- Higher-level and CAD-tool independent directives
- Directly manipulate/introspect on RTL
- Higher-level technology abstractions
What HAMMER means for this class

1. Re-use of other research tapeouts’ efforts
2. Faster flow development
3. Abstractions to reduce the complexity of VLSI flows and make them more accessible
4. Encoding designer knowledge/expertise in a robust way
   a. There’s a ton of info that ends up in people’s heads as you do stuff, and it’s hard to write stuff down in a productive way
   b. Reducing pain for future tapeout students like yourselves
Note: in real life chips need to go on boards with packages, etc.
Chisel -> FIRRTL

- Recap: Chisel is a HCL embedded in Scala
- That is to say - every Chisel design is a Scala program, which when executed, emits a concrete instance of a circuit in FIRRTL.
- We are using digital top (place and route tool will manage the top level), so we will instantiate analog macros in the digital top.
- A brief note on scan chains: we will use a scan chain generator written in Chisel
In Chisel, we specify memories using an abstract Mem()/SyncReadMem() construct:

class SRAMTest extends Module {
  val io = IO(new Bundle {
    val in  = Input(UInt(32.W))
    val en = Input(Bool())
    val out = Output(UInt(32.W))
  })
  val counter = Counter(1024)
  val mem = SyncReadMem(1024, UInt(32.W))
  when (io.en) {
    mem.write(io.in, counter.value)
    counter.inc()
  }
  io.out := mem.read(io.in)
}
circuit HelperSRAMTest :
   module HelperSRAMTest :
      input clock : Clock
      input reset : UInt<1>
      output io : {flip in : UInt<32>, flip en : UInt<1>, out : UInt<32>}

      reg value : UInt<10>, clock with : (reset => (reset, UInt<10>("h00")))
      smem mem : UInt<32>[1024]
      when io.en :
         write mport _T_10 = mem[bits(io.in, 9, 0)], clock
      [...]  
      node _T_16 = bits(io.in, 9, 0)
      read mport _T_17 = mem[_T_16], clock
      io.out <= _T_17

What it looks like in Verilog (i.e. giant bank of flip flops):
reg [31:0] mem [0:1023];
MacroCompiler

- However, by default, these memories would compile to standard cell flip-flops, which is very area-inefficient for implementing memories in contrast to SRAM macros. Example:
MacroCompiler

- Solution: FIRRTL compiler passes that identify the generic memories from Chisel/FIRRTL (ReplSeqMem) and replace them with modules which use collections of BlackBox SRAM memories (MacroCompiler) given a cache of technology SRAMs.

- ReplSeqMem: Replace mem => mem_ext (create blackboxes)
- MacroCompiler: Create the mem_ext module which uses technology SRAMs
Timing and I/O Constraints

- Clock constraints - tells the tool about clock frequencies, uncertainty/jitter, etc.
  - Can also specify related clocks
- I/O constraints - specifies input and output delays, capacitances for external pins
- I/O types/cells - specifies I/O types (input, output, tri-state) and corresponding cells to drive pins
Bumps vs Wirebond Pads

- **Bumps**: metal (e.g. Cu) bumps on top of the chip which we flip over and bond to a substrate/board
- **Wirebond pads**: wires are used to bond exposed metal on top of the chip to a substrate/package/board
- In this class, we will use wirebond pads
False Paths

- False paths
  - A logically impossible path that appears with a naive analysis.
  - Look at the timing report and declare it as a false path.
  - Dangerous if misused
Synthesis

- Maps RTL (Verilog) to a post-synthesis netlist (structural Verilog).
- Standard cells come in different sizes and drive strengths.
- The synthesis tool uses the previously-mentioned constraints to select standard cells appropriately.
- Synthesis will also perform optimizations to simplify the RTL.
  - E.g. if all of a module’s inputs are constants, it may optimize away the module entirely by precomputing its outputs.
module adder (a, b, c);
    input [1:0] a, b;
    output [1:0] c;
    wire [1:0] a, b;
    wire [1:0] c;
    wire n_0, n_1, n_3, n_4, n_5, n_6;
    NAND2X54_P0 g80__7837(.A (n_6), .B (n_5), .Z (c[1]));
    NAND2X3_P0 g81__7557(.A (n_3), .B (n_4), .Z (n_6));
    OR2X8_P0 g82__7654(.A (n_4), .B (n_3), .Z (n_5));
    NAND2X54_P0 g84__8867(.A (n_0), .B (a_0), .Z (c[0]));
    XNOR2X6_P0 g83__1377(.A (b[1]), .B (a[1]), .Z (n_3));
    NAND2AX3_P0 g86__3717(.A (a[0]), .B (b[0]), .Z (n_1));
    NAND2AX3_P0 g85__4599(.A (b[0]), .B (a[0]), .Z (n_0));
    AND2X8_P0 g87__3779(.A (b[0]), .B (a[0]), .Z (n_4));
endmodule
Floorplanning

- Recall: RTL says what to put (logic, state, macros) but doesn’t say where to put stuff
- Floorplanning is the art of specifying placement constraints.
- Main types of placement constraints:
  - Chip size - tells the place and route tool how large the chip is and how much padding there is
  - Module placement - tells the place and route tools to put cells from a certain module within a certain boundary
  - Macro placement - tells the place and route tool where to put macros (e.g. analog blocks, SRAMs, etc)
Standard Cell Layout

- Digital layout typically uses standard cells (as opposed to fully custom layouts in analog).
- Standard cells are transistor-level implementations of CMOS logic gates.
- Typical structure of a standard cell includes power/ground rails and pins.
Standard Cell Layout

- Standard cells are assembled into layouts in tracks by placing them next to each other.
- Signals are routed in layers above the standard cells.
- Power is routed to the rails (vdd and gnd) via a power plan (e.g. power grid and vias).
- Each row is typically mirrored (vdd->gnd, gnd->vdd, etc)
- Overlap rails, not abut them
Other Aspects of Floorplanning

- **Power planning** - defines the power strategy for the chip. For example, a power plan for the chip can involve creating grids for VDD and GND on each layer.
- **Tap cells** - technologies require the substrate/body to be “tapped” to a known voltage. Standard cells exist to perform this body tapping. Some stdcell architectures come with built-in taps.
- **Filler cells** - in order to meet density requirements, unused space must be filled, typically with decap.
Floorplan Visualization (Example)
Place and Route

- Given a post-synthesis netlist and floorplanning/physical design constraints, create a physical layout by placing standard cells on the chip and creating wires to route between the different cells.
- Performs standard cell placement and routing while respecting the floorplanning/physical design constraints and routing to macros (e.g. analog macros, SRAMs).
- The final result is a GDS file which can be sent to the fab.
DRC

- Design Rule Check (DRC) is the process of checking that the geometry in the GDS file follows the rules given by the fab.
- Digital standard cell layouts must still obey design rules.
- Errors often happen when designs/layouts are integrated together.
- DRC rules in advanced technologies are extremely complex and confusing.
- Sometimes CAD tools can do stupid things!

NEED SPACE?

JUST PACK THE NETS CLOSER TOGETHER. WHAT COULD GO WRONG?

(Edward Wang, June 2017)
LVS

- **Layout vs. Schematic (LVS)** is the process of checking that the geometry/layout matches the schematic/netlist.
- CAD tools can export netlists for digital designs.
- As before, LVS errors can often arise when blocks are integrated together.
- They are confusing since a shorted net can mess up the entire check!

(Lydia Lee, June 2017)
Verification

- We can run simulations on post-synthesis and post-place and route netlists (RTL) in order to check that the system still functions as intended.
- In industry, they run these checks with timing annotations so that setup and hold times aren’t violated.