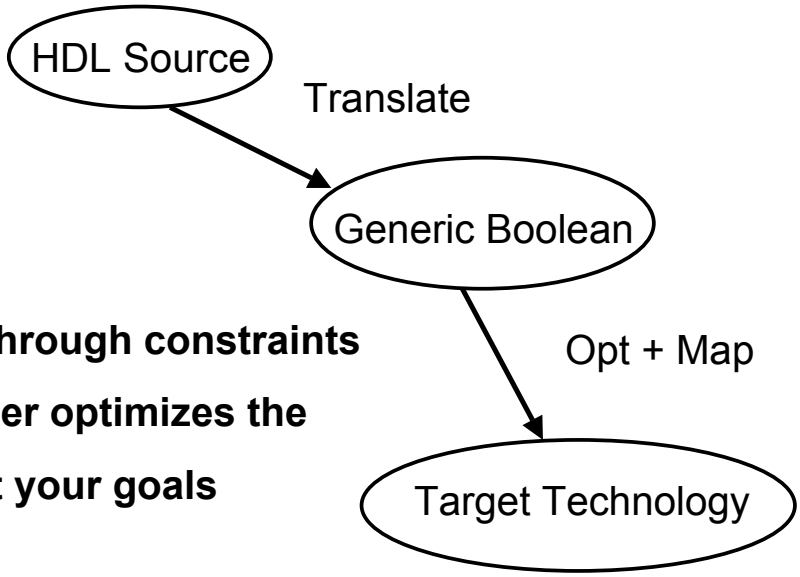
Synthesis is …



Set the goals through constraints

Design Compiler optimizes the design to meet your goals

Other information needed for logic synthesis

- What else does the synthesis tool need besides RTL?
- Library of cells into which to map
- Information about clocks
- Information about inputs
- Information about outputs
- Hints for synthesis tools

Target Standard Cell Library

- Used by Design Compiler for building a circuit
- During mapping, DC will
  - Choose functionally-correct gates from this lib
  - “time” the circuit using supplied timing data for these gates
- Produced either by fab or third parties
  - Often available for “free” (fab adds fee to wafer cost)
- Large set of combinational cells
  - Primitive: INV,NAND, NOR, AOI, OAI, XOR, MUX, etc.
  - Compound: AND,OR, ADD, etc.
  - Sequential, flip-flop/latch, {positive, negative} D-type Flip-flop with {set, reset}

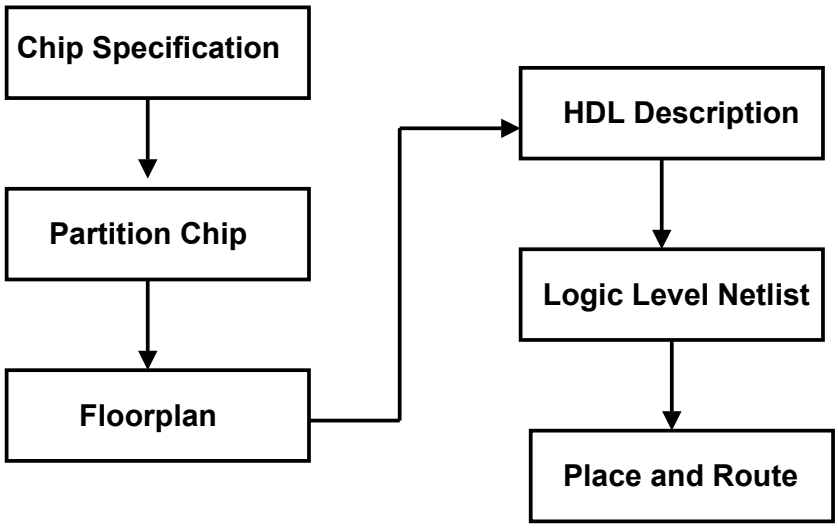
Needed for each Cell

- Logical: relationship between outputs and inputs
- Timing/Electrical:
  - Capacitance information for each pin
  - Delay information for each input->output path
  - Slope of outputs
  - Power consumption of cell
  - Required setup time for sequential cells
  - Models are generally tables
- Physical:
  - Layout
  - Abutment and Pin location for place and rout
  - Area of cell for synthesis

Partitioning for Synthesis

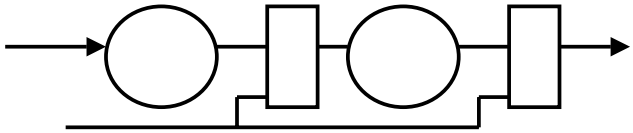
- Why?
  - Separate distinct functions
  - Workable size and complexity
  - Design reuse
  - Meet physical constraints
  - Good for team projects
- Bad partitioning may end up with long wires and slow design, no opt cross the boundaries
- Related CL and destination registers are grouped into one block
- Balance block size with run time
- Separate core logic, pads, clocks, …

Chip Synthesis Process



Coding for Synthesis

- Think Hardware!
  - Think of topology implied by the code
- Think RTL! (Register Transfer Level)
- Writing in an RTL coding style means describing
  - The register architecture
  - The circuit topology
  - The functionality between registers
- Design Compiler optimizes the logic in between not the register placement



Logic Synthesis Example – 4-bit Gray Counter

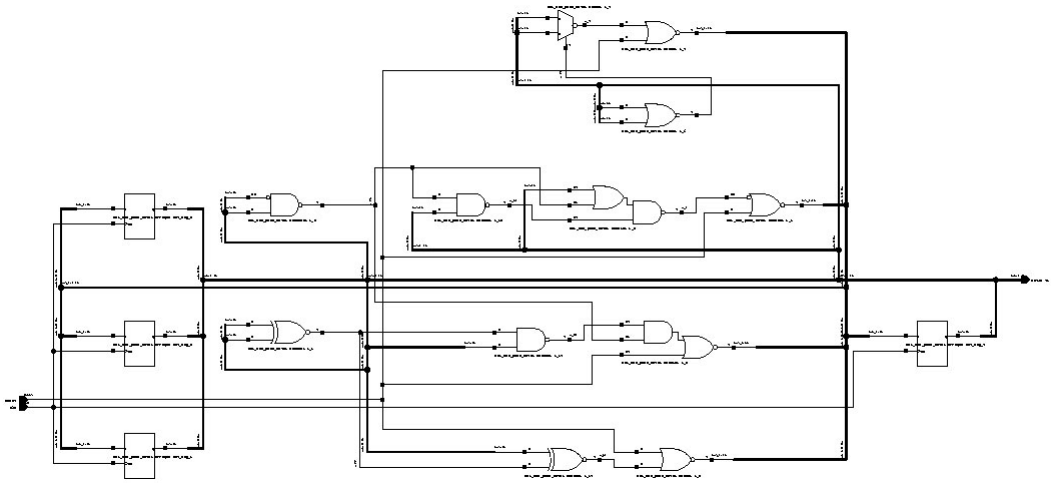
```
module gray(clk, reset,out);

input clk, reset;
output [3:0] out;

wire clk,reset;

reg [3:0] out;

always @(posedge clk)
begin
     if(reset == 1) out = 4'b0000;
     else begin
          case(out)
                    4'b0000: out = 4'b0001;
                    4'b0001: out = 4'b0011;
                    4'b0010: out = 4'b0110;
                    4'b0011: out = 4'b0010;
                    4'b0100: out = 4'b1100;
                    4'b0101: out = 4'b0100;
                    4'b0110: out = 4'b0111;
                    4'b0111: out = 4'b0101;
                    4'b1000: out = 4'b0000;
                    4'b1001: out = 4'b1000;
                    4'b1010: out = 4'b1011;
                    4'b1011: out = 4'b1001;
                    4'b1100: out = 4'b1101;
                    4'b1101: out = 4'b1111;
                    4'b1110: out = 4'b1010;
                    4'b1111: out = 4'b1110;

          endcase
     end
end
endmodule
```

Synthesis of Gray Counter

- Use 0.13μ library
- Set clock period to 2ns
- Set clock skew to 100ps
- Set input drive equal to INVX1
- Set input arrival to be 1ns after clk
- Set output load to be 4 NAND2X2 gates
- Require output valid by 1ns after clock

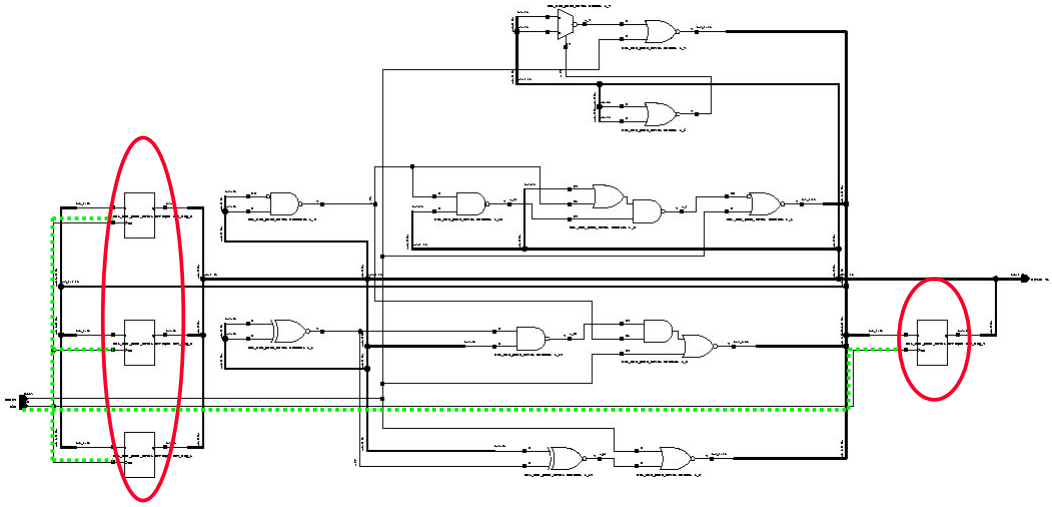
```
module gray(clk, reset,out);

input clk, reset;
output [3:0] out;

wire clk,reset;

reg [3:0] out;

always @(posedge clk)
begin
     if(reset == 1) out = 4'b0000;
     else begin
          case(out)
                    4'b0000: out = 4'b0001;
                    4'b0001: out = 4'b0011;
                    4'b0010: out = 4'b0110;
                    4'b0011: out = 4'b0010;
                    4'b0100: out = 4'b1100;
                    4'b0101: out = 4'b0100;
                    4'b0110: out = 4'b0111;
                    4'b0111: out = 4'b0101;
                    4'b1000: out = 4'b0000;
                    4'b1001: out = 4'b1000;
                    4'b1010: out = 4'b1011;
                    4'b1011: out = 4'b1001;
                    4'b1100: out = 4'b1101;
                    4'b1101: out = 4'b1111;
                    4'b1110: out = 4'b1010;
                    4'b1111: out = 4'b1110;

          endcase
     end
end
endmodule
```


Results of Synthesis



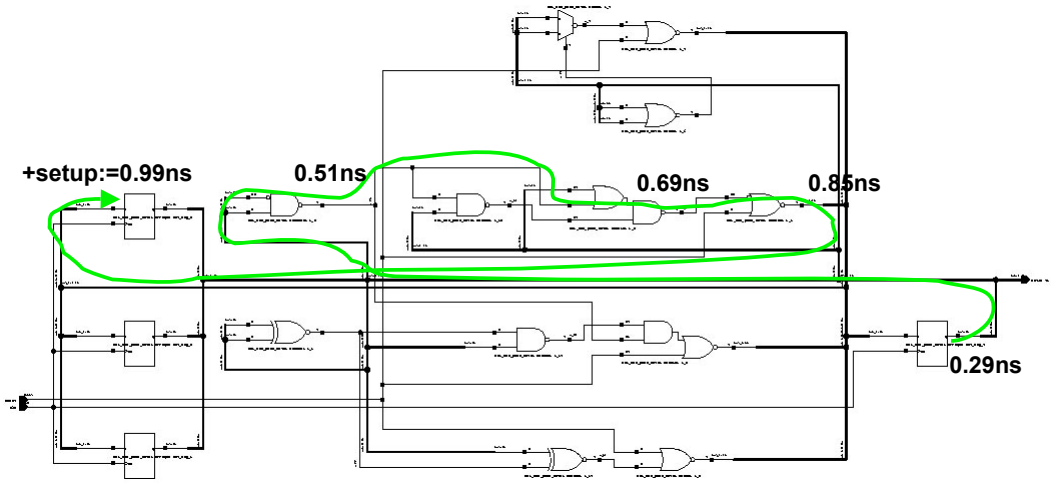
- Read back into schematic system after synthesis
- Gate placement in schematic is somewhat random

Results of Synthesis – State and Clocking



- 4-FF's – corresponds to original code
- All connected to global clock

Results of Synthesis – Critical path



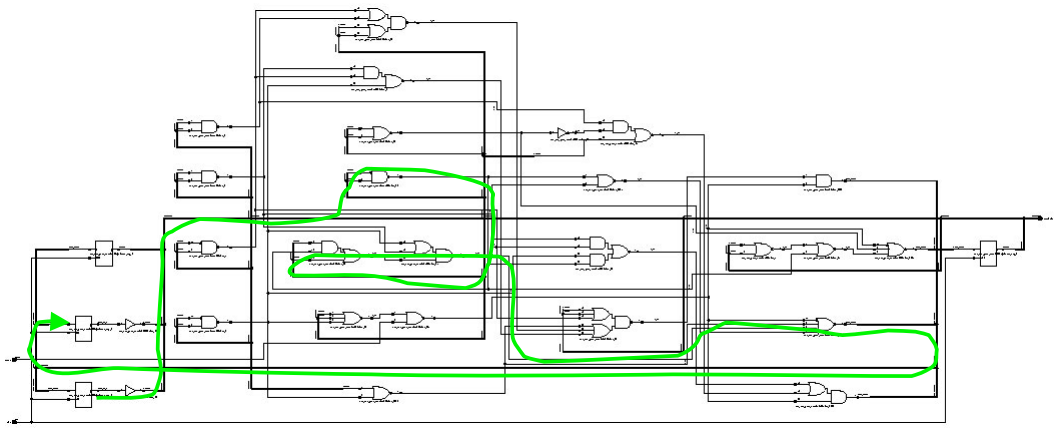
- Tool chose gates to meet required cycle time (2ns)
- Plenty of margin

Code Style and Synthesis Results

- Suppose we make the counter this way:

```
module gray(clk, reset,out);

input clk, reset;
output [3:0] out;

wire clk,reset;

reg [3:0] out;

always @(posedge clk)
begin
     if((reset == 1) || (out == 4'b1000)) out = 4'b0000;
           else if (out == 4'b0000) out = 4'b0001;
           else if (out == 4'b0001) out = 4'b0011;
           else if (out == 4'b0010) out = 4'b0110;
           else if (out == 4'b0011) out = 4'b0010;
           else if (out == 4'b0100) out = 4'b1100;
           else if (out == 4'b0101) out = 4'b0100;
           else if (out == 4'b0110) out = 4'b0111;
           else if (out == 4'b0111) out = 4'b0101;
           else if (out == 4'b1001) out = 4'b1000;
           else if (out == 4'b1010) out = 4'b1011;
           else if (out == 4'b1011) out = 4'b1001;
           else if (out == 4'b1100) out = 4'b1101;
           else if (out == 4'b1101) out = 4'b1111;
           else if (out == 4'b1110) out = 4'b1010;
           else if (out == 4'b1111) out = 4'b1110;
end

endmodule
```


If-else Gray Counter



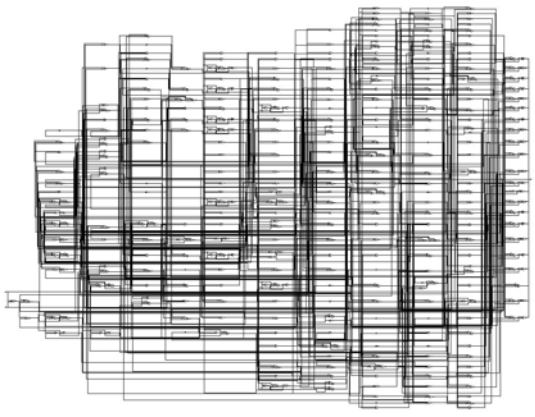
- 26 gates instead of 12
- Slowest path is 1.4ns instead of 1ns

if-else Statements and *case* Statements

- *If-else* Implies multiplexing hardware
- To infer latches, use *if* without an *else* clause
- *If-then-elseif* statements **imply priority**
  - Use only if priority checking is required
  - Otherwise esult will be incorrect or possibly slower
- *case* Statements imply parallel mux function
  - Use case statements where possible, particularly for FSM's

Combinational Adder Contents

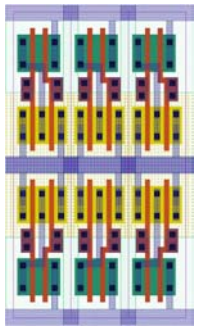
- 369 cell instances from 1 line of verilog
- Did you really need it?



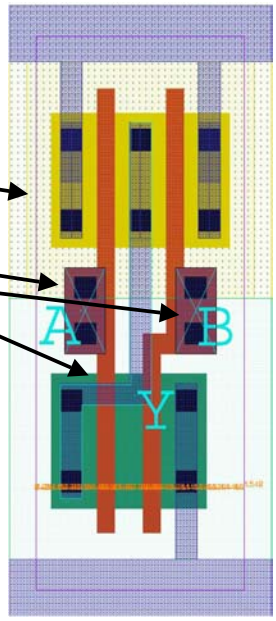
- Tool created 32-bit Carry Lookahead Adder
  - Tool contains parameterizable set of arithmetic operators
  - Grabs the appropriate one for your probem

Layout for Std Cells

- Mask data – for fab
- Abutment - for placement
- Pin location – for routing
- Area of cell – for synthesis



Cells arrayed to abutment line



2-Layer metal Place and Route (Old Style)

- Arrange cells in rows interspersed with routing channels
- Routing runs vertically in m1, horizontally in m2
- No routing over cells

Placement for Multiple Levels of Routing

- No need to leave channels
- Most (80-90%) of surface can be covered by cells