* Primarily targets digital design (though analog extensions exist)
* Multiple Levels of abstraction
  + Behavioural
    - Sequential behaviour (Serial)
      * o/p of 1 functional block to input of another
    - Parallel Behaviour
      * Can pass a block o/p to i/p's of a number of blocks
      * Separate, simultaneous events
  + Structural
    - Hardware Component Hierarchy
      * Eg: hierarchical block diagrams, component netlist
    - Software Subroutine Hierarchy
* Timing
  + Needs to model propagation delays, clock period, timing checks eg setup and hold time, clock to queue delay
  + Supports simulation of estimated design timing
* Levels of Abstraction
  + Behavioural
    - Mathematical equations, algos, flow graphs, pseudo code
    - Can omit timing
    - Could be in the form of:
      * Boolean expressions/truth tables
      * FSM's
      * High level algo
    - Used by block architects for max simulation speed and flexibility in modelling the architecture
    - Least detail -> fast design entry, fast simulation
    - Used for test benches
    - Divide by 2 example:  
      always @(din)  
         dout = din/2;
  + RTL (Data Path Design)
    - Nets and registers
    - Partition system into combinational and sequential logic
      * using constructs and coding styles supported by logic synthesis
    - Define timing in terms of cycles based on clocks
    - Buses, Registers, multiplexers, decoders, adders, multipliers, etc
    - Independent of implementation tech
    - Used by block implementors
    - Used by hardware designers for synthesis
    - Used for synthesizable code
    - Divide by 2 example:  
      always @(posedge clk)  
        dout <= din >> 1 //non-blocking assignment <=

* Gate/Structural Level
  + Built-in and user-defined primitives
  + Instantiate and interconnect predefined components (vendor provided macrocells or built-in logic primitives)
  + Gates, flip flops (standard cells)
  + Used by library developers for physical level tools
  + AKA netlist
  + Conflicting requirements during optimization
    - Minimizing gates
    - Minimizing gate levels (delay)
    - Minimizing signal level transitions (power)
  + Divide by 2 example:

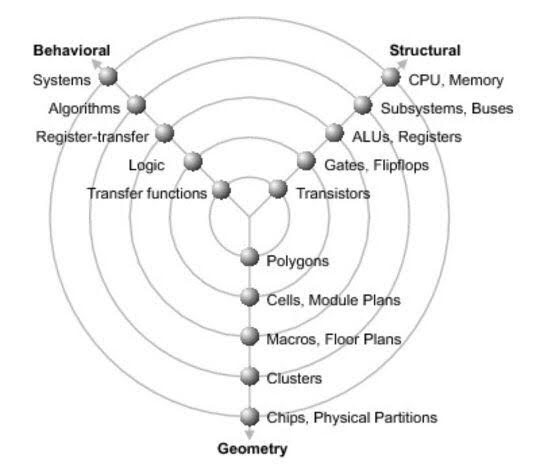
FD1 op[3:0] (  //instance array op of cell type FD1  
  .D( {1'b0, din[3:1]} ),  
  .CP(clk), .Q(dout) );

* Switch Level
  + Using built in switch primitive (NMOS, PMOS Transistors)
  + Transistors
  + Used by library developers for place and route tools
  + Most detailed -> slow design entry and simulation

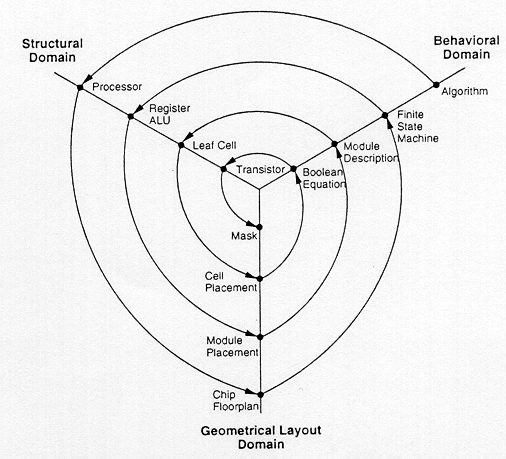
* Layout Level
  + Large number of geometric shapes corresponding to different fabrication layers
  + Final target may be FPGA
* Why HDL?
  + Allows user to abstractly capture design intent
  + ASCII text - quickly capture and modify design
  + Can design higher abstraction levels - find errors earlier in the design
  + Enables design reuse (using previous code or IP providers)
  + RTL is independent of technology
    - Can defer choice of target (ASIC/FPGA, etc)
    - Can switch technology/description
    - Can make architectural/functional changes
    - Adapt design to future projects
* HDL Based Simulation
  + At various levels - logic level, switch level, circuit level
  + Self-checking test benches
  + Vendor independent design

* HDL used by::
  + System architects: high-level architectural exploration
  + Verification engineers: test benches to test components and systems
  + HW designers: RTL code for synthesis
  + Model developers: system-level IP and ASIC/FPGA macrocells

* Formal Verification
* Testability Analysis and Test Pattern Generation
* Gajski-kuhn Y chart:



* More practical/realistic representation (top down design flow):



* Adoption Issues
  + No standard design methodology
  + Primarily targets digital design
  + Planning and partition required before coding
  + Coding style influences time to closure