ASIC Logic

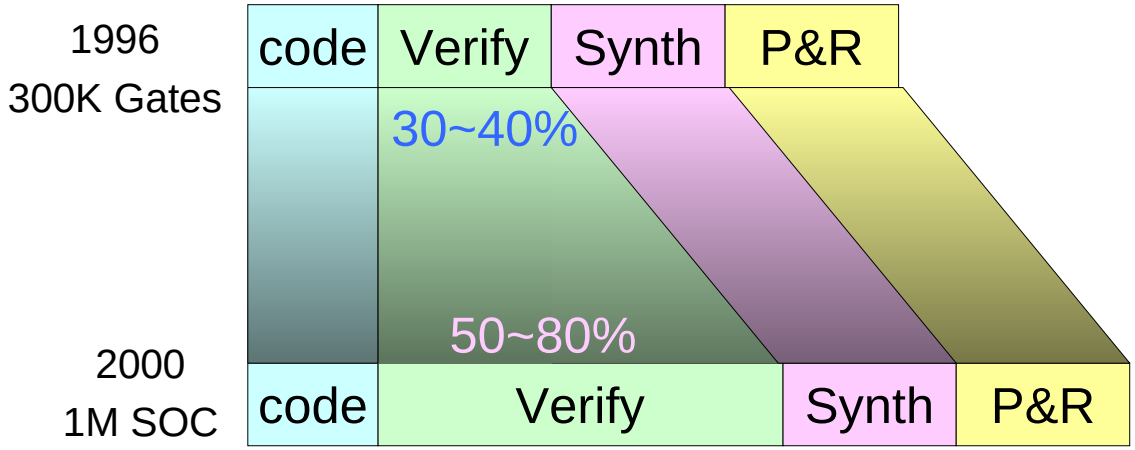
Speaker: Hao-Yun Chin

Advisor: Prof. **Tian-Sheuan Chang**

Apr. 20, 2004

SoC Verification

- Problem: 50~70% of design process is spent in verification



- An effective verification methodology is highly desirable

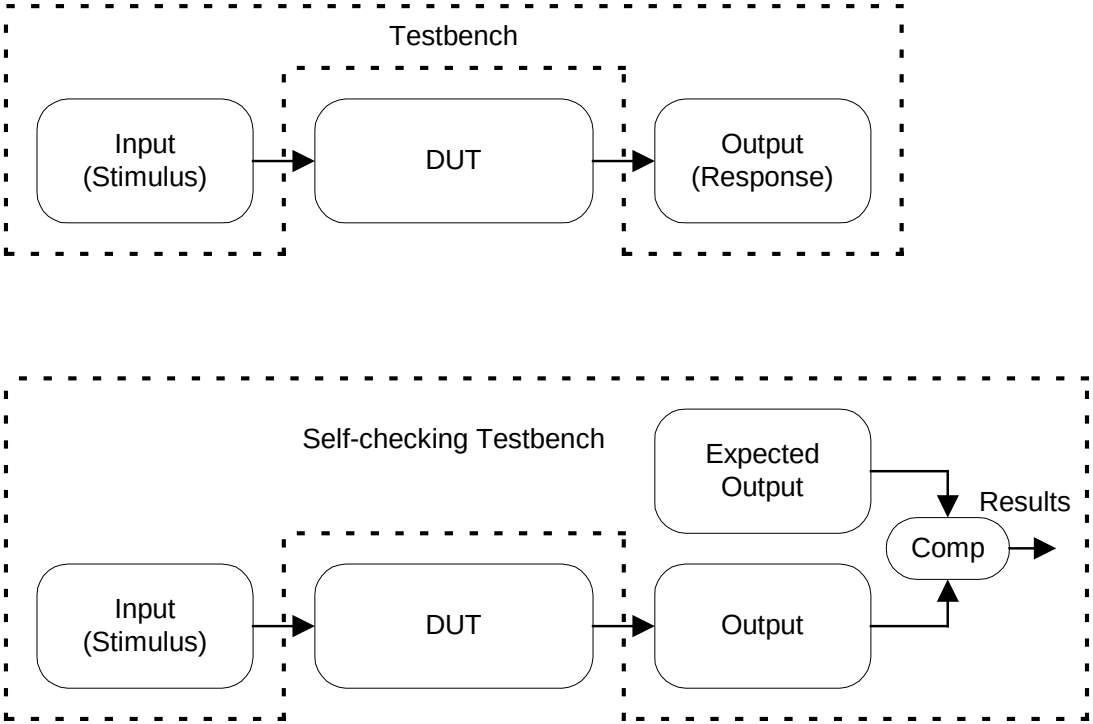
Verification Technology Overview

- Simulation Technology
  - Event-based
  - Cycle-based
  - Transaction-based
  - Code coverage
  - HW/SW co-verification
  - Emulation
  - Rapid prototyping
  - Hardware accelerator
- Static Technology
  - Lint check
  - Static timing
- Formal Technology
  - Theorem proving
  - Formal model check
  - Formal equivalence check

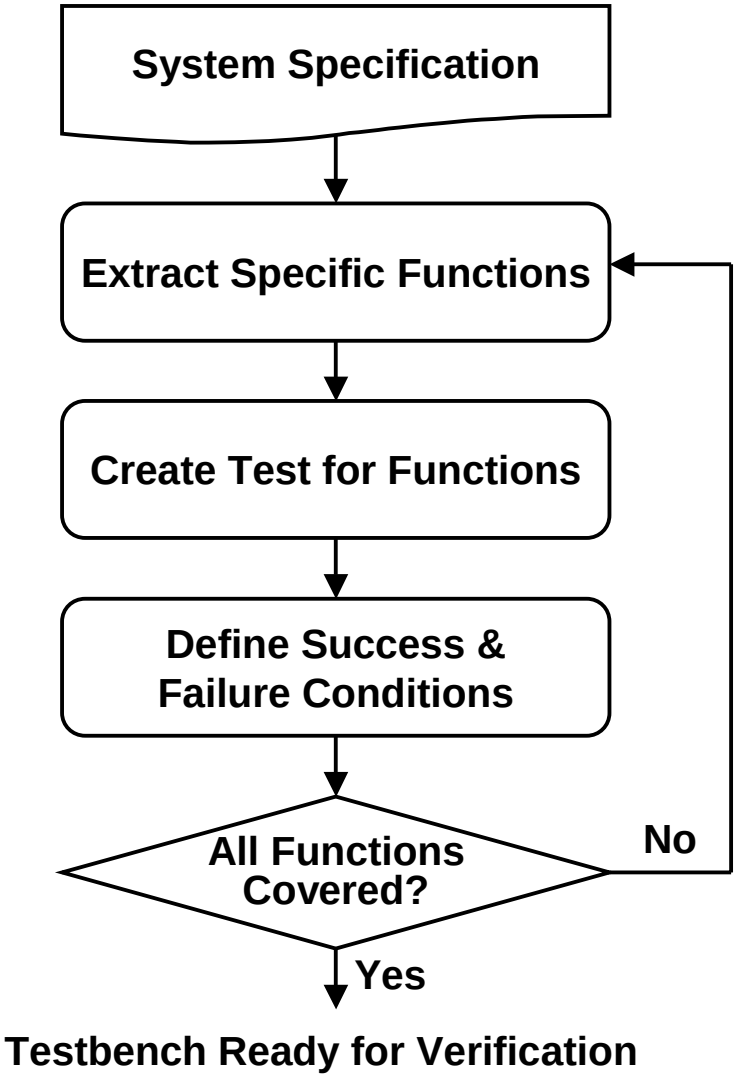
Coverage-Driven Verification

- Quantitatively analyze the simulation completeness with well-defined coverage metrics
  - Although 100% coverage still cannot guarantee a 100% error-free design
- Generate more patterns for the uncovered areas using formal techniques or designers' knowledge
- Tests optimization by eliminating tests that do not add new coverage
- Prioritize tests for regression runs
- Provide a more systematic way to manage the verification process

Testbench


Testbench
Input
(Stimulus)
DUT
Output
(Response)
Self-checking Testbench
Expected
Output
Results
Comp
Input
(Stimulus)
DUT
Output

Testbench Creation



Focus on:

- Corner Cases
- Boundary Conditions
- Design Requirements
- Error Conditions
- Exception Handling

Types of Coverage

- Code coverage
  - Statement coverage
  - Block coverage
  - Decision coverage
  - Path coverage
  - Expression coverage
  - Event coverage
  - Toggle coverage
  - Variable coverage
- FSM coverage
  - Conventional FSM coverage
  - Semantic FSM (SFSM) coverage
- Functional Coverage

Table	Types of verification coverage
Coverage type	**Alternate names**
Statement execution	Line, statement, block, basic block, segment
Decision	Branch, all edges
Expression	Condition, condition-decision, all edges, multiple condition
Path	Predicate, basis path
Event	(None)
Toggle	(None)
Variable	(None)
State machine	State value, state transition, state scoring, variable transition, FSM

Statement Coverage

$$Statement\ \text{coverage}\ \% = \frac{Number\ \text{of statements executed}}{Total\ \text{number of executable statements}} \times 100$$

```
always @ ( in or reset ) begin
❶ out = in;
❷ if ( reset ) ❸out = 0;
❹ en = 1;
end
```

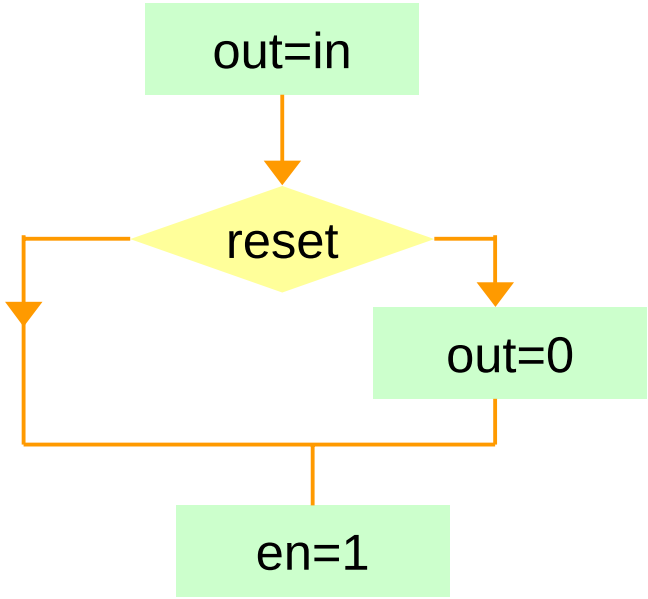
There are ***4*** independent statements.

Branch Coverage

$$Branch \text{ coverage } \% = \frac{\text{No. of program branches taken}}{Total \text{ no.of possible branches in the HDL}} \times 100$$

Measure the coverage of each branch in the *if* and *case* statements

```
always @ (in or reset) begin
    out = in;
    if ( reset )    out = 0;     else ?
    en = 1;
end
```



Implied *else* is also measured.

Differences between SC and BC

Design

If (b==a)

  c=1;

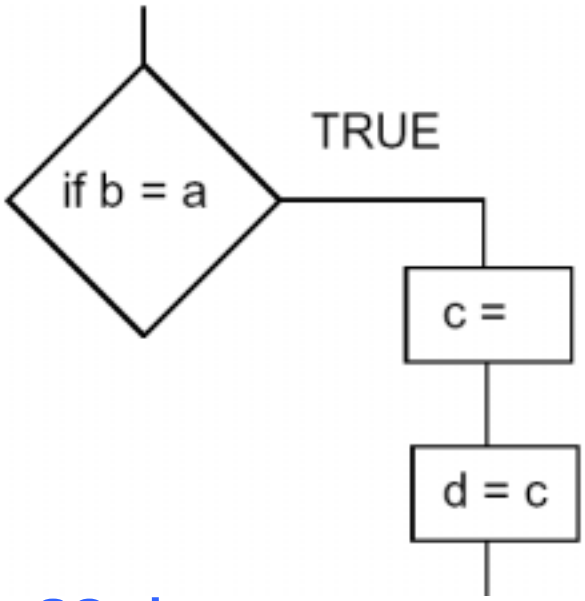
d=c;

In simulation

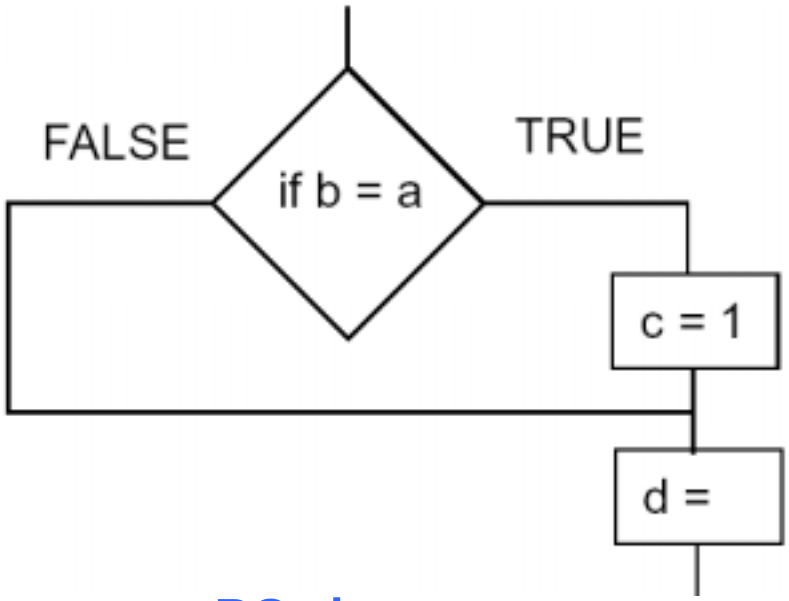
b is forced to always equal a

SC = 100%

BC = 50%



SC view



BC view

Typical Coverage Targets

Measurement	Coverage Test (%)
Statement	100
Branch	100
Condition	60~100 *
Path	> 50
Toggle	100

* Depending on coverage tool

FSM Coverage

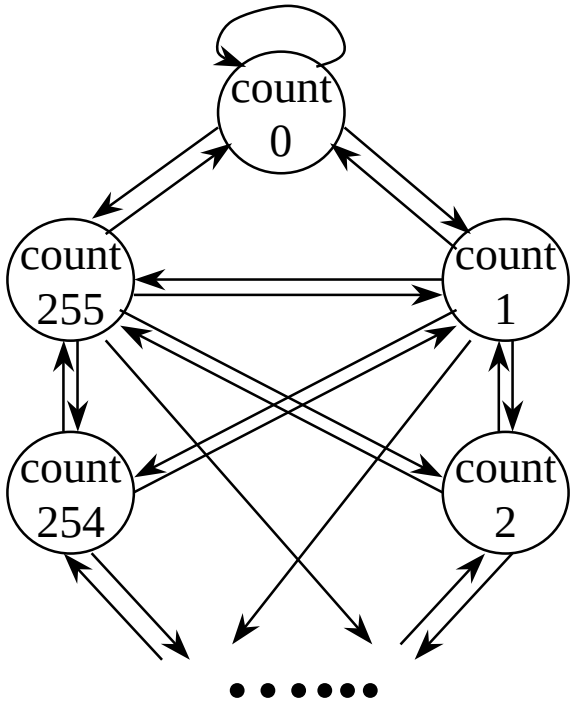
- State
- Arc
  - An arc is a transition between two 'adjacent' states.
  - The arc coverage metric reports on those arcs actually traversed during simulation, expressing these as a proportion of all possible arcs defined in the HDL code.
- Path
  - Identifies all the fundamental **cyclic** paths from which it then constructs one or more **supercycles** which represent the main functionality of the FSM. The smaller cycles are then a part of the supercycles.
  - To the extent to which supercycles represent the intended operation modes of the FSM, a measure of coverage can then be obtained by:
    - The percentage of all supercycles that have been fully traversed
    - The number of times a particular subordinate cycle has been traversed

Conventional FSM Coverage

- The measurement of state visitation and state transitions

```
module counter (clk, rst, load, in, count) ;
input          clk, rst, load ;
input   [7:0] in ;
output [7:0] count ;
reg      [7:0] count ;

always @(posedge clk) begin
    if (rst) count = 0 ;
    else if (load) count = in ;
    else if (count == 255) count = 0 ;
    else count = count + 1 ;
end
endmodule
```



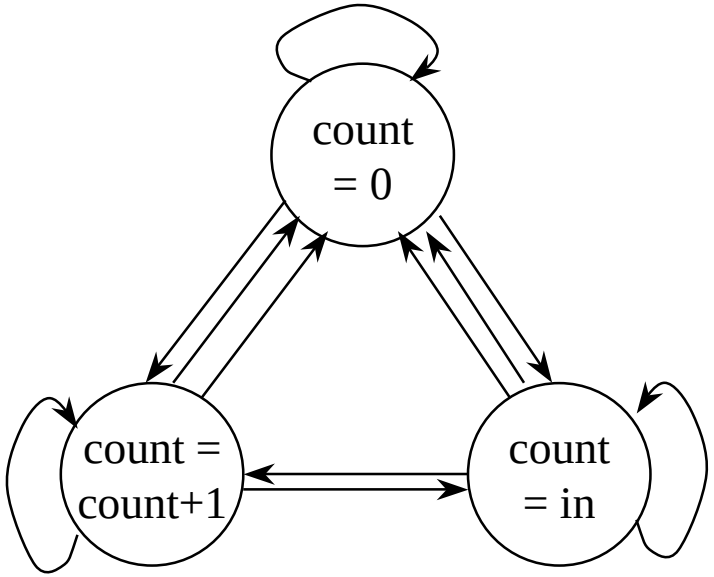
256 states 66047 transitions

Semantic FSM Coverage

- Merge the states with same behavior into one semantic state to reduce the complexity

```
module counter (clk, rst, load, in, count) ;
input          clk, rst, load ;
input   [7:0] in ;
output [7:0] count ;
reg      [7:0] count ;

always @(posedge clk) begin
    if (rst) count = 0 ;
    else if (load) count = in ;
    else if (count == 255) count = 0 ;
    else count = count + 1 ;
end
endmodule
```

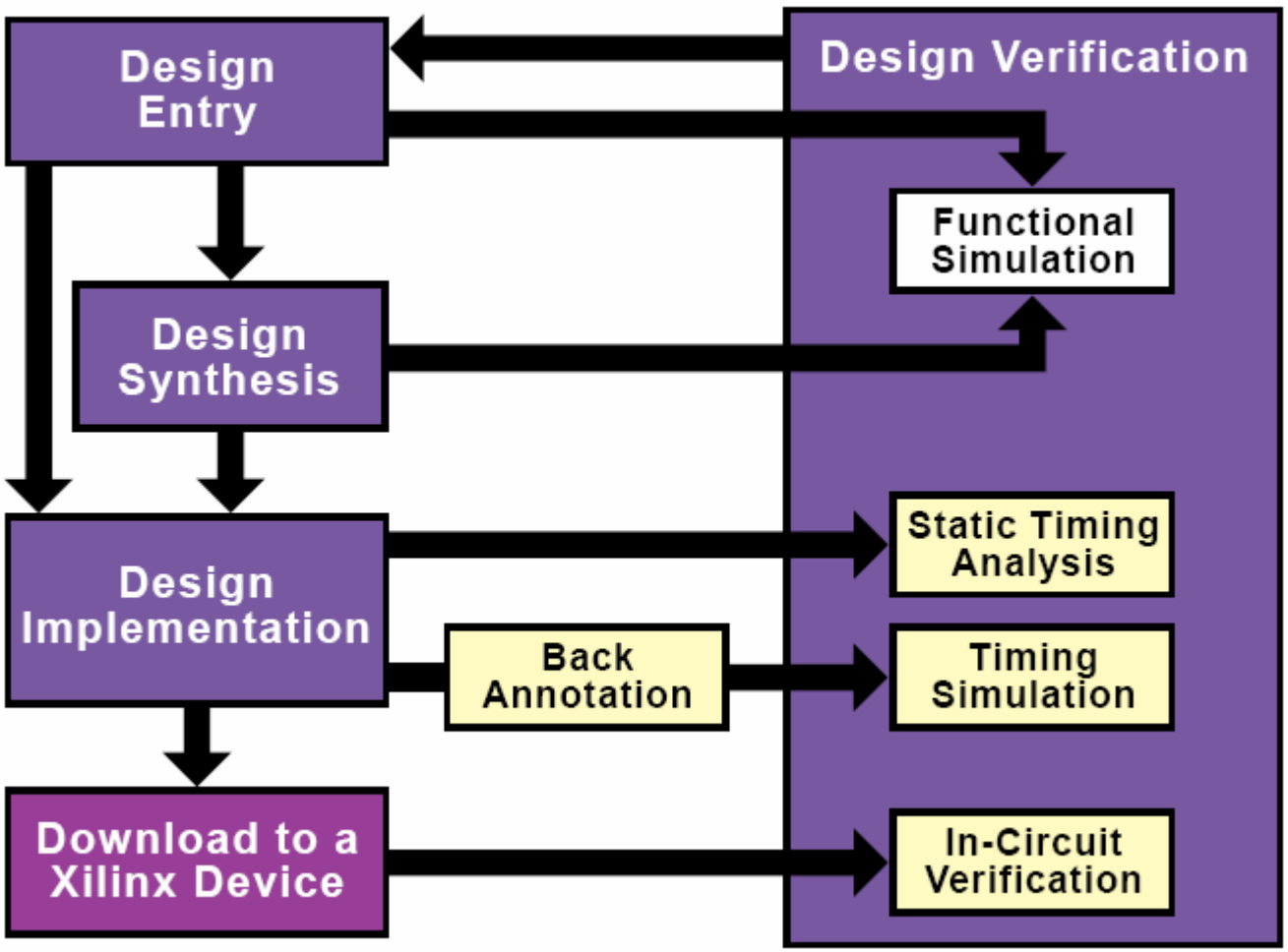


3 states 11 transitions

More Verification Information

- Verification Methodology Manual, 3rd Edition
  Techniques for Verifying HDL Designs
  Author: David Dempster and Michael Stuart
  - http://www.dacafe.com/DACafe/EDATools/BOOKINFO/TransEDA/index.html
  - Worked examples for TransEDA VN in Appendix

Xilinx ISE Design Flow

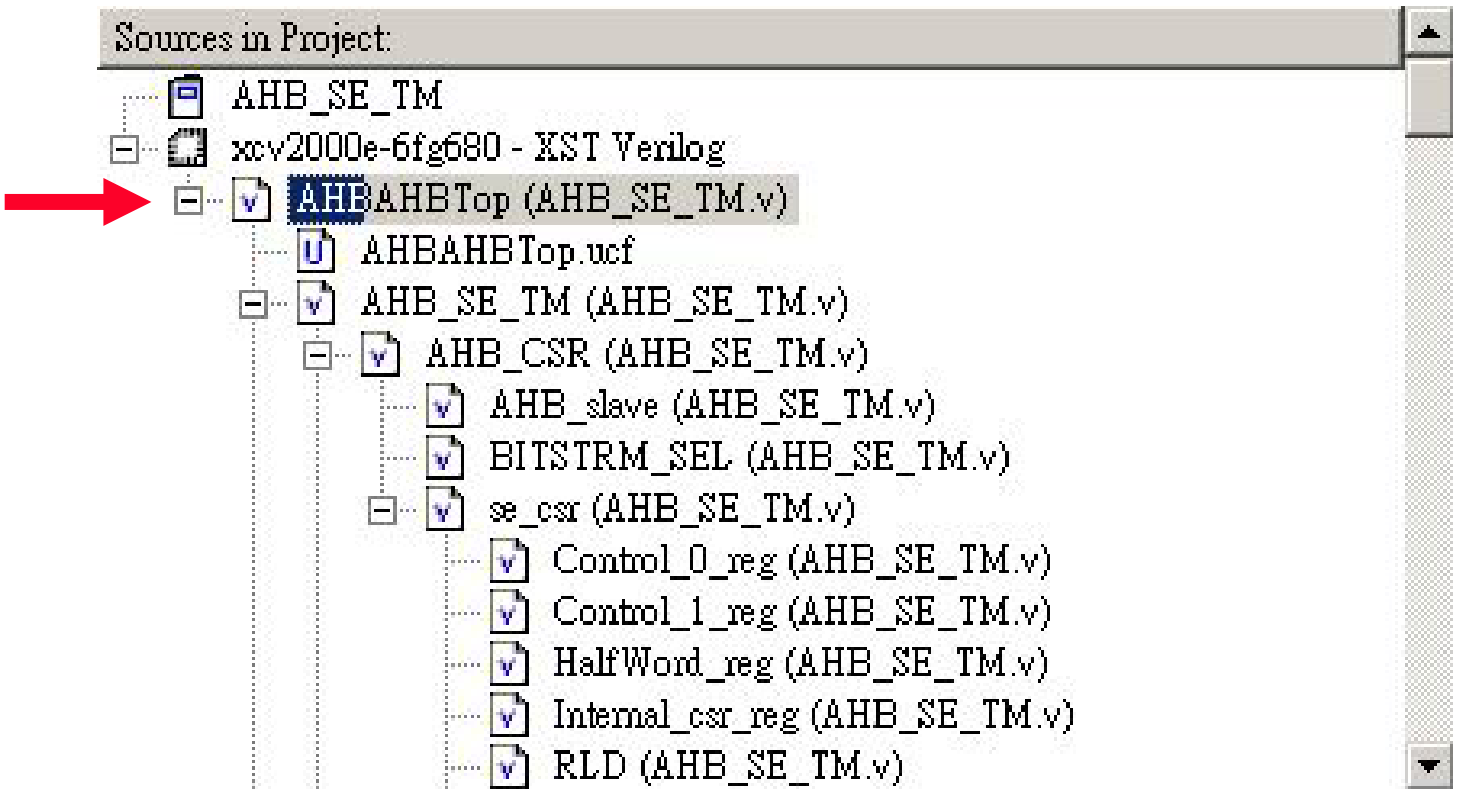


Xilinx Design Flow

1	Synthesis	Synthesis
2	Translate	Implement Design
3	Map	
4	Place & Route	
5	Trace	Generate Post P&RTiming
6	Generate Bitstream	Generate Programming File

Xilinx ISE

- Select Top Module Before You Run Any Process



Xilinx ISE

- Run All Processes
  - V is great
  - ! Is ok, but check report file
  - X means sorry, wrong design
- Report the post P&R static timing of your design
  - An ARM7 runs at 20Mhz typically
  - So, try to make your design run at 20+ Mhz