CSE 45493 - 3

Verification Plan

Verification Plan

- This is the specification for the verification effort. It indicates what we are verifying and how we are going to do it!

Role of the Verification Plan

- Traditional approach
  - Do as you wish!
- New approaches
  - Want metrics to determine when verification is done
  - Plan specifies the verification effort
  - Defines 1st time success (ideally)

Specifying the Verification

- When are you done?
  - Metrics
- The specification determines “what has to be done”!
- The specification does NOT determine the following:
  - How many will it take?
  - How long will it take?
  - Are you are doing what needs to be done?
- Verification Plans help define these!

Specifying the Verification (continued)

- The plan starts from the design specification
  - Why? The reconvergence model!!
- The plan must exist in written form
  - "the same thing as before, but at twice the speed, with these additions…" – Not acceptable specification.
- The specification is a golden document – it is the "law".
- It is the common source for the verification efforts and the implementation efforts.

Defining 1st time success

- The plan determines what is to be verified.
  - If, and only if, it is in the plan, it will it be verified.
  - The plan provides the forum for the entire design team to determine 1st time success.
  - For 1st time success, every feature must be identified and under what conditions. As well as expected response.
- Plan documents these features (as well as optional ones) and prioritized them.
  - This way, consciously, risk can be mitigated to bring in the schedule or reduce cost.

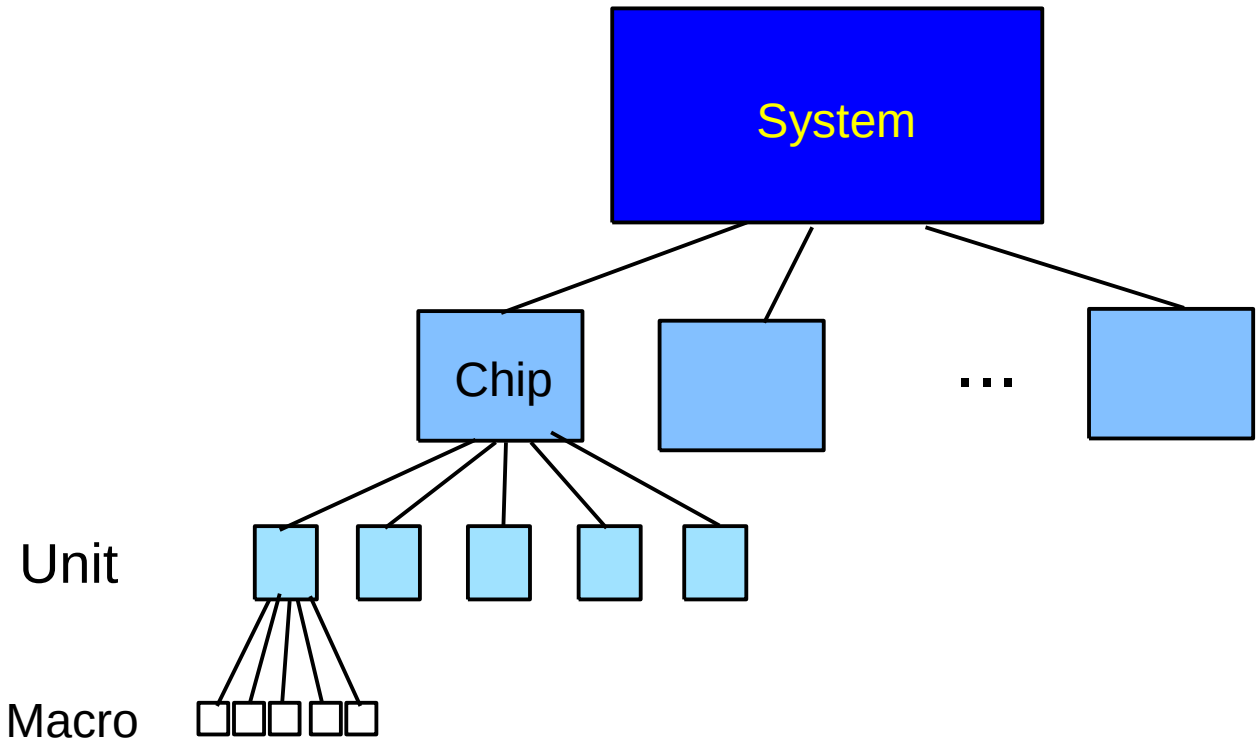
Defining 1st time success (cont.)

- Plan creates a well defined line, that when crossed can endanger the whole project and its success in the market.
- It defines:
  - How many test scenarios (testbenches/testcases) must be written
  - How complex they need to be
  - Their dependencies
- From the plan, a detailed schedule can be produced:
  - Number of resources it will take – people, machines, etc
  - Number of tasks that needs to be performed
  - Time it will take with current resources

Verification plan

- Team owns it!
  - Everyone involved in the project has a stake in it.
  - Anybody who identifies deficiencies should have input so they are fixed.
- The process used to write the plan is not new. It has been used in the industry for decades. It is just now finding its way to hardware. Others who use it:
  - NASA
  - FAA
  - Software

Hierarchical design



Allows design team to break system down into logical and comprehendible components. Also allows for repeatable components.

Verification levels

- Verification can be performed at various granularities:
  - Designer/Unit/sub-subunit
  - ASIC/FPGA/Reusable component
  - System/sub-system/SoC
  - board

Verification levels (continued)

- How to decide what levels? There are trade-offs:
  - Smaller ones offer more control and observability but more effort to construct more environments
  - Larger ones, the integration of the smaller partitions are implicitly verified at the cost of lower control and observability but less effort because fewer environments
- Whatever level is decided, those pieces should remain stable.
- Each verifiable piece should have its own specification document

Designer level

- Usually pretty small
  - One design of a design engineer
- Interfaces and functionality tend to change often
- Usually don’t have independent specification associated with them
- Environment (if one) is usually ad hoc
  - Not intended to gain full coverage, just enough to verify basic operation (no boundary conditions)

Unit level

- Typically designs assigned to one design engineer
- Each unit level piece is stressfully verified
  - Unlike the ad hoc approach
- If unit level is fully verified, then you only need to worry about the boundary conditions.
- High number of units in a design usually means it is not reasonable to do unit level due to high level of effort for environments

Reusable component level

- Independent of any particular use
- Intended to be used as is in many different parts of a design.
- Saves in verification time.
  - Can code BFM's for these pieces that others who communication with the reusable component can use.
- They necessitate a regression suite
- Components will not be reused if users do not have confidence in them
  - Gained by well documented and executed process

ASIC/FPGA level

- Physical partition provides a natural boundary for verification
  - Once specified, very little change due to ramification down stream.
- FPGA’s used to get debugged at integration. Now they are so dense, that a more ASIC flow is required.

System level

- What is a system?
  - Everyone's definition is a little different.
- Verification at this level focuses on interaction, not function buried in one piece.
- A large ASIC may take this approach. (Assuming that all pieces that make up the ASIC are specified, documented, and fully verified).
  - Assume that individual components are functionally correct.
  - Testcase defines the system.

Board level

- Main Objective:
  - Confirm the connectivity of board design tool
    - Perhaps better in formal verification
- Can also be used as a system to verify its functionality
- When doing board level verification, one must ensure that what is in the “system”, is what is to be manufactured.
- Things to consider
  - Problem: Getting models for all components
    - 3rd party?
  - Many board level components don’t fit into a digital simulation – analog!
  - Board-level parasitic

Current practices for verifying a system

- Designer Level Sim
  - Verification of a macro (or a few small macros)
- Unit Level Sim
  - Verification of a group of macro's
- Chip (Element) Sim
  - Verification of an entire logical function (i.e. processor, storage controller, Ethernet MAC, etc.)
- System Level Sim
  - Multiple chip verification
  - Sometimes done at a 'board' level
  - Can utilize a mini operating system

In an Ideal World….

- Every Macro has had a perfect verification
  - All permutations are verified based on legal inputs
  - All outputs checked on the small chunks of the design
- Unit, Chip, and System level would then only need to check interconnections
  - Ensure that macros used correct Input/Output assumptions and protocols

Reality

- Macro verification across a system is not feasible
  - May be over 100 macros on a chip
    - Would require 200 verification engineers
    - Shortage of skilled verification engineers
    - Business can’t support the development expense
- Verification Leaders must make reasonable trade-offs
  - Concentrate on Unit level
  - Designer level on riskiest macros

Verification Strategies

- Level of granularity
- Types of testcases for each granularity
- Test cases: white Box, Black Box, Grey Box
- Level of abstraction
  - Processor example
- How to verify the response
- How random will it be

Verifying the Response

- Deciding how to apply stimulus is usually easy part.
- Deciding response is the hard part.
  - Must plan how to determine the expected response
  - How to verify that design provided the expected response
  - Self checking testbenches is recommended
- Detect errors and stop simulation as early as possible. Why?
  - Error identified when it takes place – easier to debug
  - No wasted simulation cycles

Random Verification

- Does not mean randomly applying ‘0’s and ‘1’s
  - Inputs must produce valid stimulus
  - The sequence of operations and the content of the data transferred is random!
    - To creates conditions that one does not think of
    - Hit corner cases
    - Reduces bias from the engineer
- Very complex to specify and code
- Must be a fair representation of operating conditions
- More complicated to determine expected response
- With proper constraints, a random environment can produce directed tests

Specifications to Features

- 1st step in planning is to identify verifiable features
- Label each feature with a short description
- Cross referenced specification and verification
- Unit level features – bulk of verification
- System level features
- Concentrate on functional errors, not tool induced errors
- Specify features for the proper level of verification

Features to testcases

- Prioritize the features
  - Must have features gate tape out
  - Less important feature receive less attention
  - This helps project managers make informed decisions when schedule pressures arise.
- Group features; become a testcases; assign to an engineer
- Describe Testcases and cross-reference to features list
- If a feature does not have a testcase, not being verified
- Define dependencies – helps prioritize
- Specify testcase stimulus (or constraints for random)
- Also must specify how each feature will be checked
- Inject errors to ensure that checks are working

Testcases to testbenches

- Group like testcases into testbenches
  - Similar configuration or same abstraction level
- Cross-reference testbenches with testcases
- Assign each group of testcases to a verification engineer
- Verifying the testbench
  - Use a broken model for each feature to be verified
  - Peer reviews
  - Provide logging messages in the testbench

Verification Do's and Don'ts

- Do:
  - Talk to designers about the function
  - Spend time thinking about all the pieces that need to be verified and situations that designer might have missed
  - Talk to "other" designers about the signals that interface to DUV
- Don't:
  - Rely on the designer's word for input/output specification
  - Allow a chip to go out without being properly verified – pay now or pay later (with inflation!)

Terminology

- Facilities: a general term for named wires (or signals) and latches. Facilities feed gates (and/or/nand/nor/invert, etc) which feed other facilities.
- EDA: Engineering Design Automation--Tool vendors. IBM has an internal EDA organization that supplies tools. We also procure tools from external companies
- Behavioral: Code written to perform the function of logic on the interface of the design-under-test
- Macro: A behavioral ; A piece of logic
- Driver: Code written to manipulate the inputs of the design-under-test. The driver understands the interface protocols.

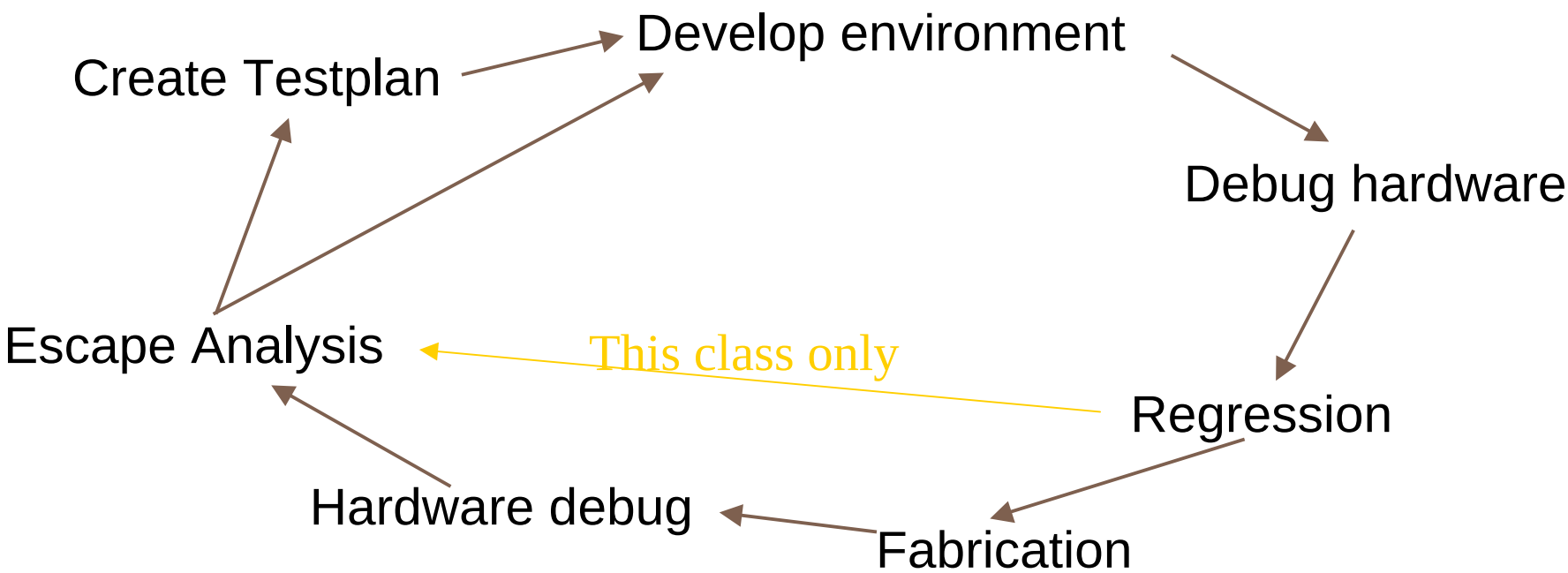
Terminology (Cont.)

- Checker: Code written to verify the outputs of the design-under-test. A checker may have some knowledge of what the driver has done. A check must also verify interface protocol compliance
- Snoop/Monitor: Code that watches interfaces or internal signals to help the checkers perform correctly. Also used to help drivers be more devious
- Architecture: Design criteria as seen by the customer. The design's architecture is specified in documents and the design must be compliant with this specification
- Microarchitecture: The design's implementation
- Microarchitecture refers to the constructs that are used in the design, such as pipelines, caches, etc.

Escape analysis

- Escape analysis is a critical part of the verification process
- Fully understand bug! Reproduce in sim if possible
  - Lack of repeat means fix cannot be verified
  - Could misunderstand the bug
- Why did the bug escape simulation?
- Update process to avoid similar escapes in future (plug the hole!)

Verification Cycle



Enterprise Systems Group Hardware Design Process

Customer

High Level Design/ System Structure

High Level Chip Design

Substrate/MCM Design

VHDL Chip Implementation

Microcode Design

SE Code Design

Logic Verification

Chip Design Team

Chip Timing, Circuit Design, Integration

Manufacturing

You are here

Hardware Fabrication

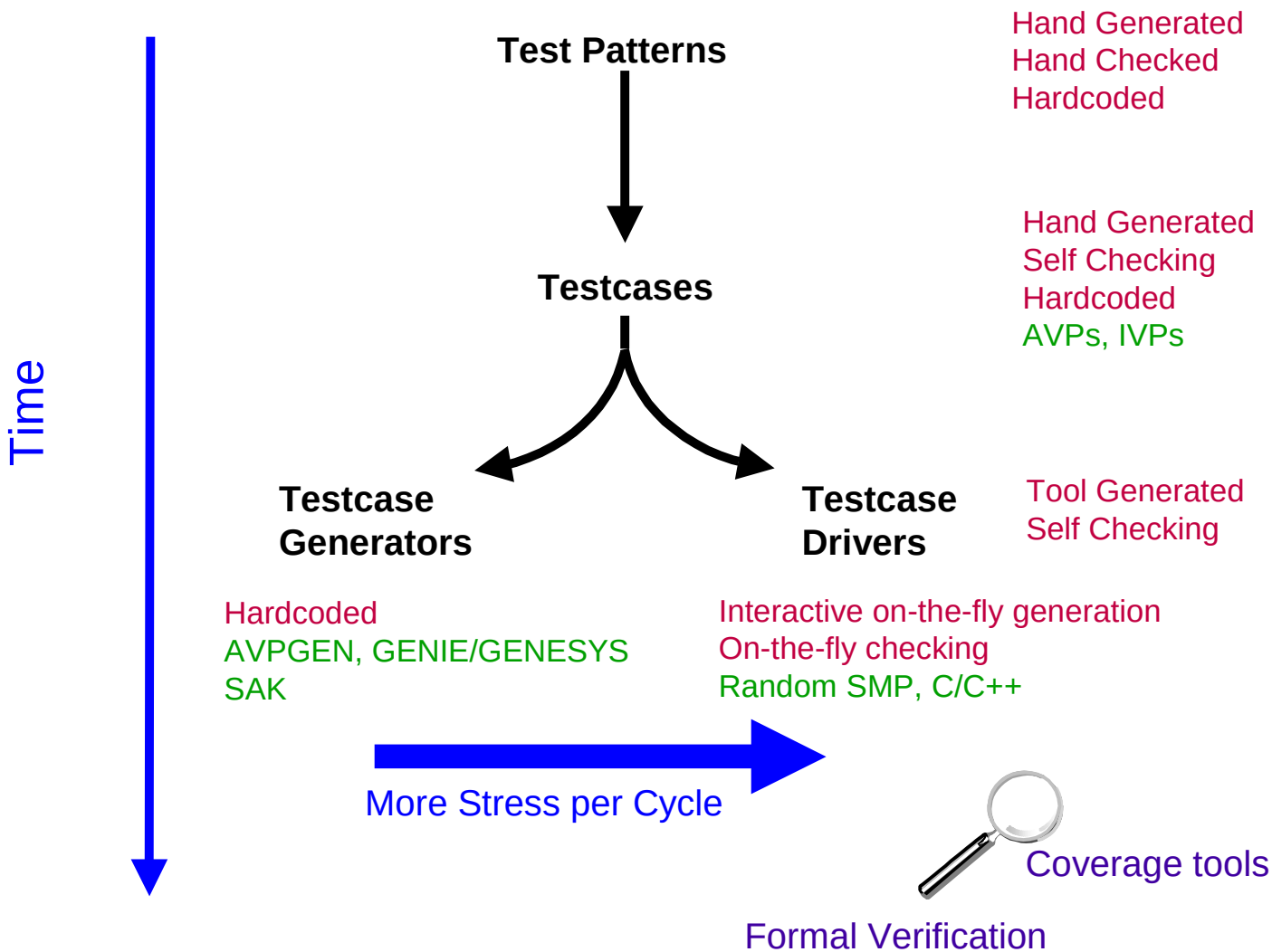
Systems Test

Bruce Wile
8/1/99

Testplan summary

- Testplan includes
  - Schedule
  - Specific tests and methods by simulation level
  - Required tools
  - Input criteria
  - Completion criteria
  - What is expected to be found with each test/level
  - What's not covered by each test/level

Verification Methodology Evolution

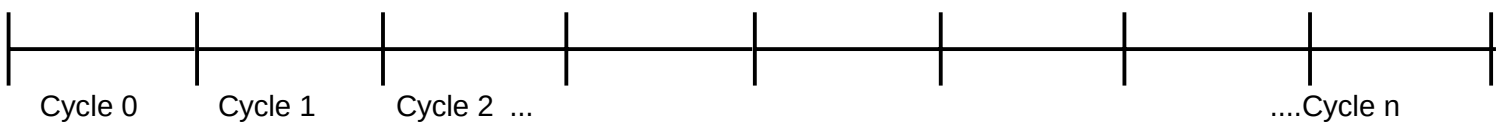


Escape Analysis: Classification

- We currently classify all escapes under two views
  - ➔Verification view
    - What areas are the complexities that allowed the escap
    - Cache Set-up, Cycle dependency, Configuration dependency, Sequence complexity, and expected resul
  - ➔Design View
    - What was wrong with the logic?
    - Logic hole, data/logic out of synch, bad control reset, wrong spec, Bad logic

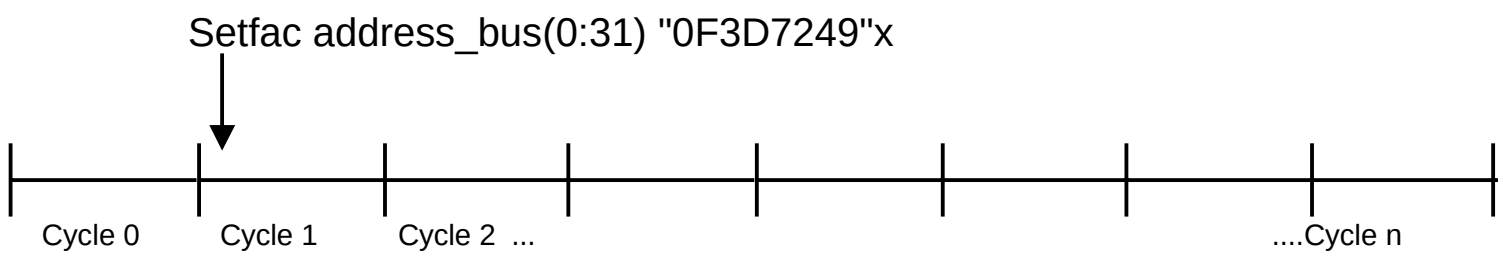
Basic Testcase/Model Interface: Clocking

- Clocking cycles
  - ➔A simulator has the concept of time.
    - Event sim uses the smallest increment of time in the tar technology
    - All other sim environments use a single cycle
  - ➔A testcase controls the clocking of cycles (movement of time)
    - All APIs include a clock statement
    - Example: "Clock(n)", where n is the increment to clock (usually '1')



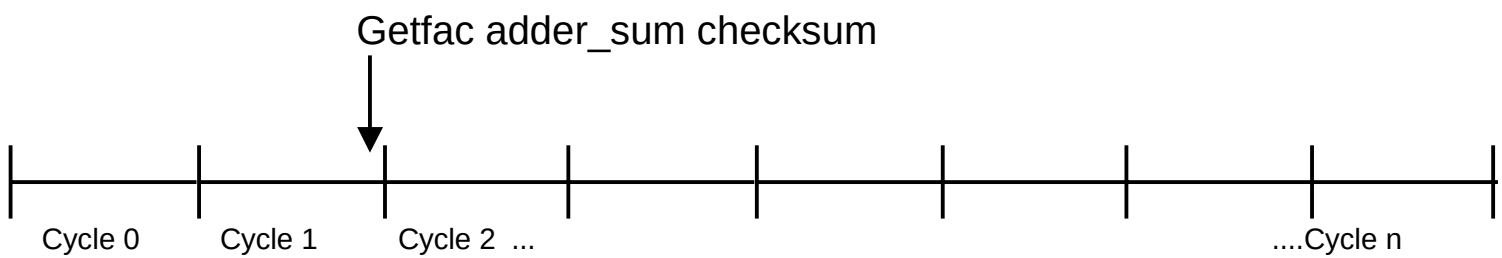
Basic Testcase/Model Interface: Setfac/Putfac

- Setting facilities
  - ➔A simulator API allows you to alter the value of facilities
  - ➔Used most often for driving inputs
  - ➔Can be used to alter internal latches or signals
  - ➔Can set a single bit or multi-bit facility
    - values can be 0,1, or possibly X, high impedence, etc
  - ➔Example syntax: "Setfac facility_name value"



Basic Testcase/Model Interface: Getfac

- Reading facilities values
  - ➔A simulator API allows you to read the value of a facility
  - ➔Used most often checking outputs
  - ➔Can be used to read internal latches or signals
  - ➔Example syntax: "Getfac facility_name varname"



Basic Testcase/Model Interface: Putting it togeth

- Clocking, setfacs and putfacs occur at set times during a cycle
  - ➔Setting of facilities must be done at the beginning of the cycle.
  - ➔Getfacs must occur at the end of a cycle
  - ➔In between, control goes to the simulation engine, where t logic under test is "run" (evaluated)