ENCS5337: Chip Design Verification

Spring 2023/2024

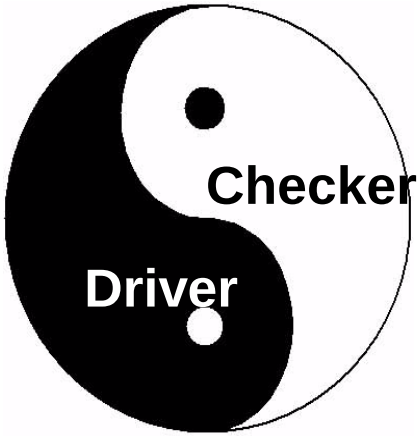
Verification Hierarchy

Dr. Ayman Hroub

Many thanks to Dr. Kerstin Eder for most of the slides

Outline

- Observability and Controllability
- Verification hierarchy
  - Levels of verification
- Fundamentals of Simulation-based Verification:
  - Strategy
    - Driving principles
    - Checking strategies



Observability and Controllability

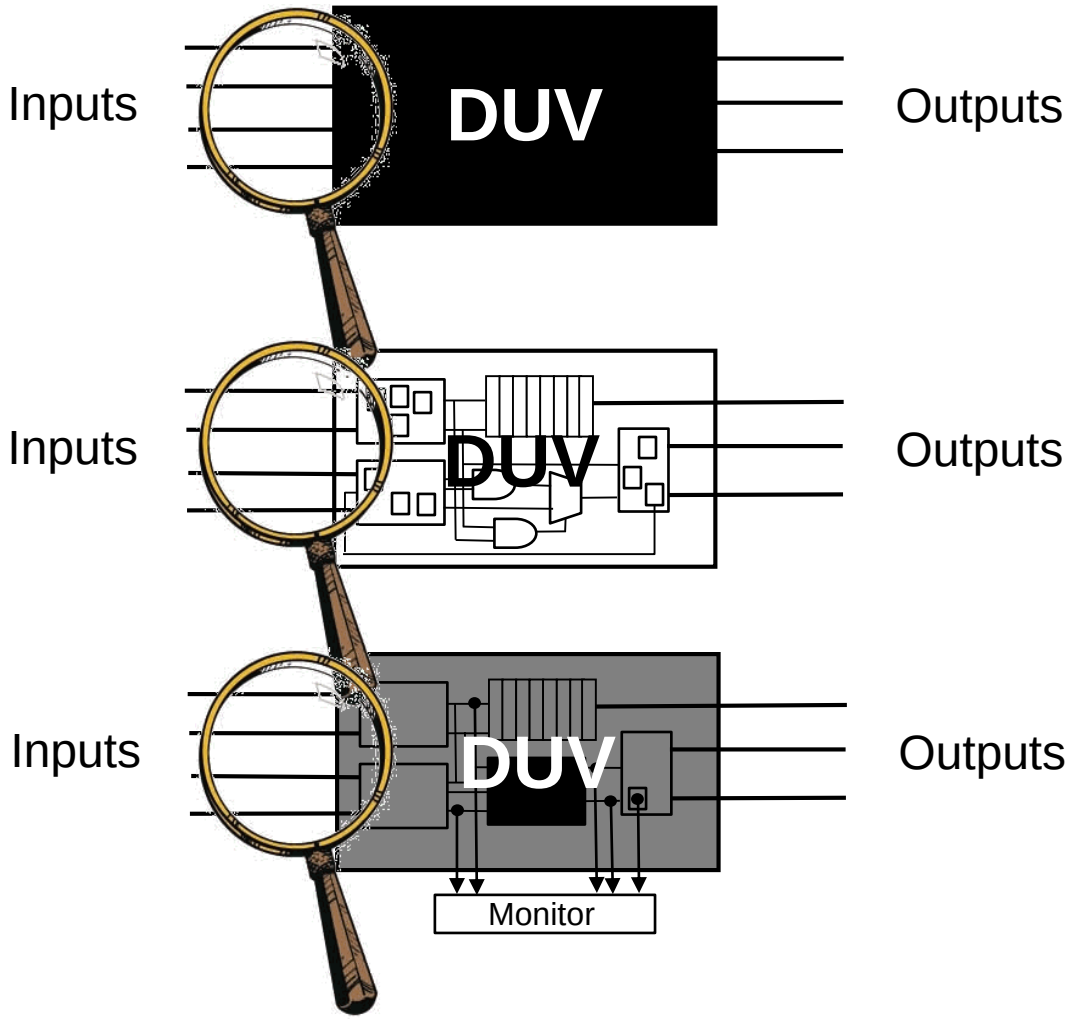
Observability and Controllability

- Observability: Indicates the ease at which the verification engineer can identify when the design acts appropriately versus when it demonstrates incorrect behavior.

- Controllability: Indicates the ease at which the verification engineer creates the specific scenarios that are of interest.

Levels of Observability

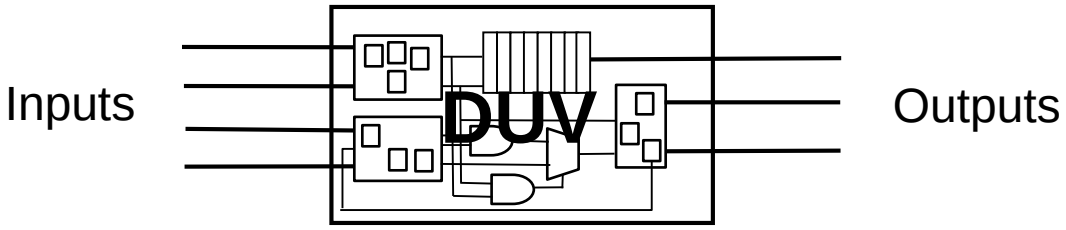
- Black Box
- White Box
- Grey Box



Black Box Verification

- The black box has inputs, outputs, and performs some (well documented) function.
- To verify a black box, you need to **understand the function.**
- The verification code utilizes only the external interfaces.
- The internal signals and state remain in the dark.
- **Pros:**
  - No knowledge of the actual implementation is required.
  - Ability to predict functional results based on inputs alone ensures that the reference model remains independent from the DUV implementation.
  - Verification code is less sensitive to changes inside the DUV.
- **Cons:**
  - Difficult to locate source of problem, only exposes effects. (If at all! Remember, not all bugs propagate to the outputs.)
  - **Lacks controllability and observability.**

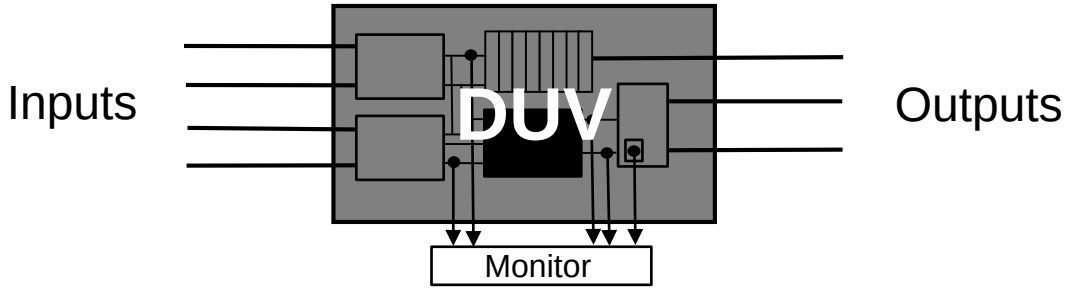
White Box Verification



(Opposite of black-box approach.)

- For white box verification the internal facilities of the DUV are known, visible and utilised for verification.
- **Pros:**
  - **Full visibility and controllability of internal signals.**
    - **Can identify and cover corner cases.**
    - **Can detect bugs as soon as they occur.**
  - Quickly possible to set up interesting conditions.
- **Cons:**
  - Danger to follow the implementation/design instead of the specification.
  - Sensitive to changes in the DUV (implementation).
  - Too many details make it hard to create and maintain.

Grey Box Verification



- For grey box verification a limited number of DUV facilities are utilised in a mostly black-box environment.
  - Access important and stable features, the rest is kept in the dark.
- Combines the pros (if done the right way) or the cons (if done the wrong way) of black and white box.
  - Progression from black box to grey box should be carefully planned and started only when the **DUV is sufficiently stable.**
- In practice: **Most verification environments are grey box.**
  - May need to start with black box with planned evolution into grey box.
  - **Note:** Prediction of correct results on an interface is occasionally impossible without viewing an internal signal.

Be careful with White Box Controllability

- In theory, the same levels as for observability also exist for controllability:
  - black, grey and white box
- In practice:
  - **We seldom control the internals of the DUV.**
  - This may drive the design into a state that is not reachable *under normal circumstances*.
  - It may thus lead to an inconsistent DUV state.
- The main exception: **Warm Loading**
  - Brings the DUV to a predefined interesting state.
    - E.g. cache initialization, almost full buffer
  - Reduces the time needed for reaching this state.

Verification Hierarchy

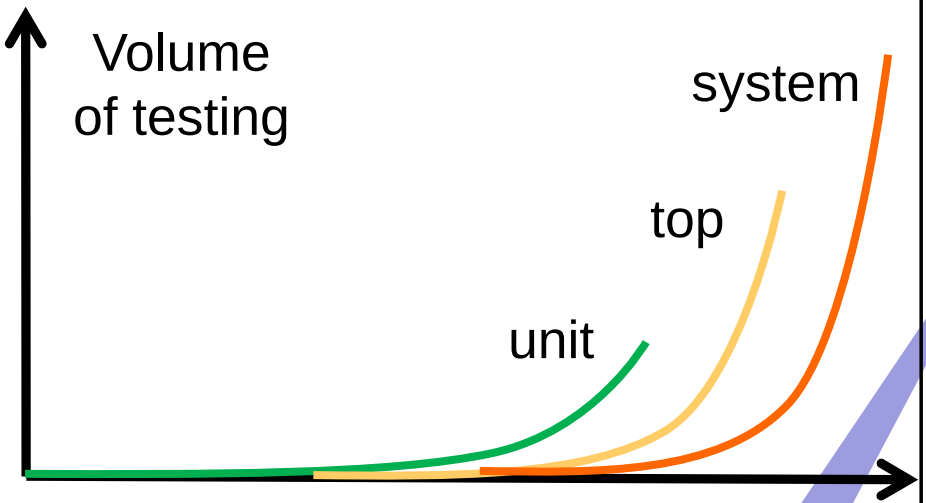
Verification Hierarchy

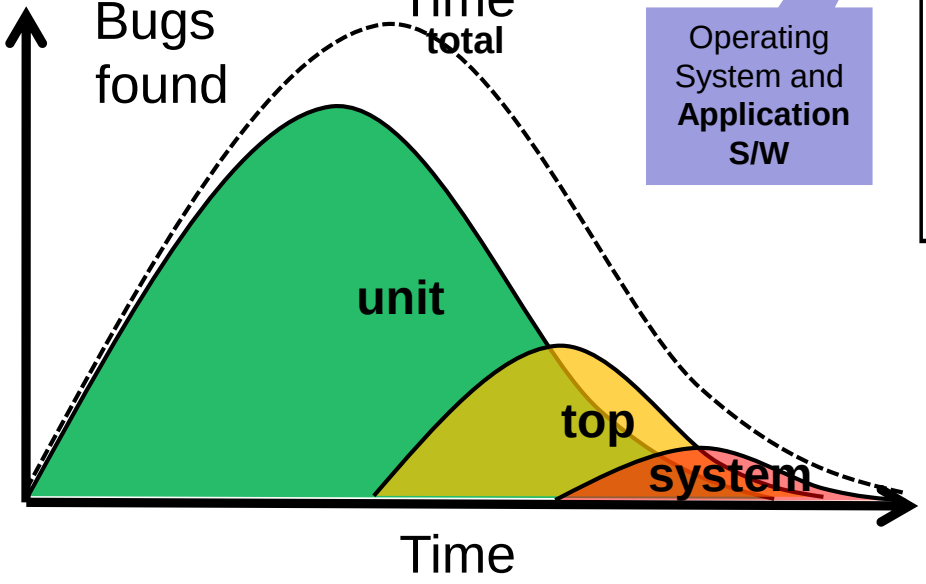
- Today's complex chips and systems are divided into logical units
  - Usually determined during specification / high-level design
  - Usually follow the architecture of the system
  - This practice is called hierarchical design
- Hierarchical design allows a designer to subdivide a complex problem into more manageable blocks
  - The design team combines these blocks to form bigger units, and continues to merge/integrate these blocks until the chip or system is complete

Pros and Cons of Hierarchical Design

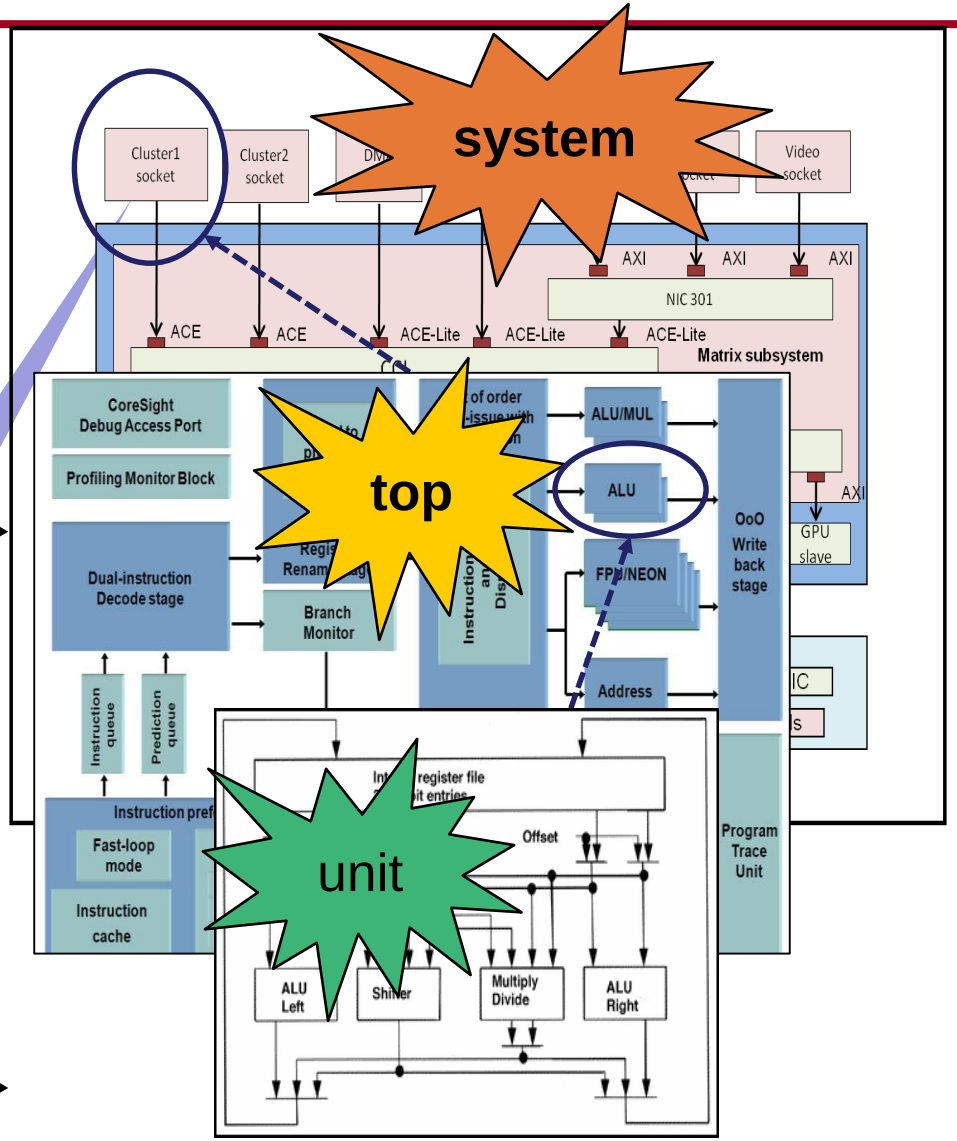
- Pros
  - Breaks the design into manageable pieces
  - Allow designers to focus on single function / aspect of the design
- Cons
  - More interfaces to specify / design / verify
  - Integration issues

Verification at different Design Levels





Operating System and **Application S/W**



Levels of Verification

- Verification usually adapts to and takes advantage of the hierarchical design stages and boundaries
- Common levels of verification
  - Designer level (block level)
  - Unit level
  - (Core level)
  - Chip level
  - System level
  - Hardware / software co-verification

Designer (Block) Level Verification

- Used for verification of single blocks and macros
- Usually, done by the designer him/herself
- Main goal – Sanity checking and certification for a given block
- Ranges from a simple test of basic functionality to complete verification environments
- The common level for formal verification

Unit Level Verification

- A set of blocks that are designed to handle a specific function or aspect of the system
  - E.g., memory controller, floating-point unit
- Usually have formalized spec
  - More stable interface and function
- The target of first serious verification effort
- Verification is based on custom-made verification environment

Core Level Verification

- A core is a unit or set of units designed to be used across many designs
  - Well defined function
  - Standardized interfaces
- Verification need to be thorough and complete
  - Address all possible uses of the core
- The verification team can use “Verification IP” for the standardized interfaces

Chip Level Verification

- Verification of a set of units that are *packaged* together in a physical entity
- Main goals of verification
  - Connection and integration of the various units
  - Function that could not be verified at unit level
- Need verification closure to avoid problems in tape-out

System Level Verification

- The purpose of this level of verification is to confirm
  - Interconnection
  - Integration
  - System design
- Verification focuses on the interactions between the components of the system rather then the functionality of each individual component

HW / SW Co-Verification

- Combines the system level hardware with the code that runs on it
- Combines techniques from the hardware verification and software testing domains
- This combination creates many issues
  - Different verification / testing techniques
  - Different modes of operation
  - Different speed
- Beyond the scope of this course

Which Level To Choose?

- Always choose the lowest level that completely contains the targeted function
- Each verifiable piece should have its own specification
- Function may dictate verification levels
  - The appropriate level of control and observability drives decisions on which levels to select for verification

Which Level To Choose?

- In general, each level that is exposed to the “outside world” is mandatory
  - For example, chip level, system level
- The rest depends on many factors
  - Complexity
  - Risk
  - Schedule
  - Resources

Fundamentals of Simulation-based Verification

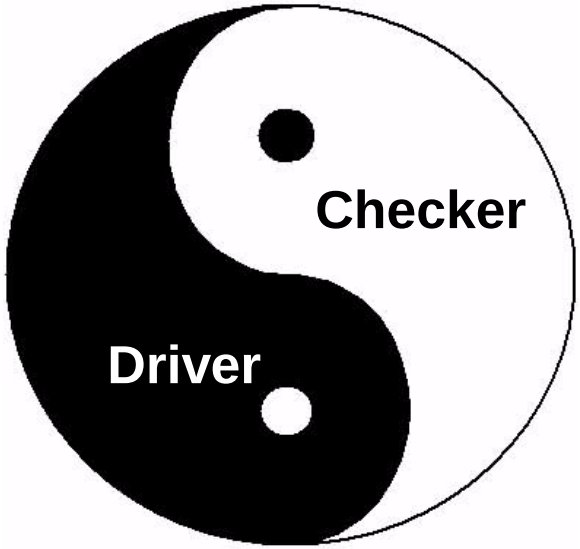
The Strategy of
Driving & Checking

Strategy of Verification

- Verification can be divided into two separate tasks
  1. Driving the design - Controllability
  2. Checking its behavior - Observability
- The basic questions a verification engineer must ask
  1. Am I driving all possible input scenarios?
  2. How will I know when a failure has occurred?

The Yin-Yang of Verification

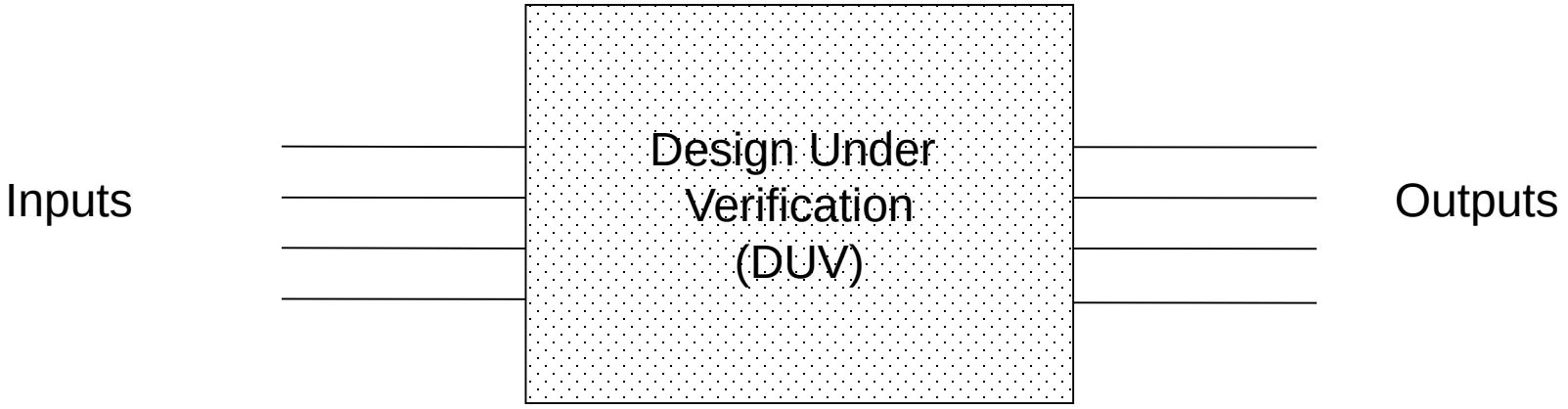
- Driving and checking are the yin and yang of verification
  - We cannot find bugs without creating the failing conditions
    - Drivers
  - We cannot find bugs without detecting the incorrect behavior
    - Checkers



Comments on Yin and Yang

- This perfect harmony does not always exist
  - Not all failing conditions are equal
    - Same bug can lead under different failing conditions to different failures (with big difference in consequences)
  - We cannot (or don't want to) detect all incorrect behaviors
    - Some are not important enough
    - For others we have safety nets
- The right balance is a function of the level of verification and specific needs
  - Example: Block vs Chip level verification – difference in drivers and checkers and in focus of verification.

The Black Box Example



- What does it mean to
  - Drive all input scenarios
  - Know when the design fails

Verification of the Black Box

- Black box since we don't look inside it
  - What does this mean?
- The black box may have a complete documentation … or not
- To verify a black box the verification engineer must
  - understand the function and be able to
  - predict the output based on the inputs.
- It is important that the verification team obtain the input, output and functional description of the black box from a source other than the HDL designer
  - Standard specification
  - High-level design
  - Other designer that interfaces with the black box
  - …

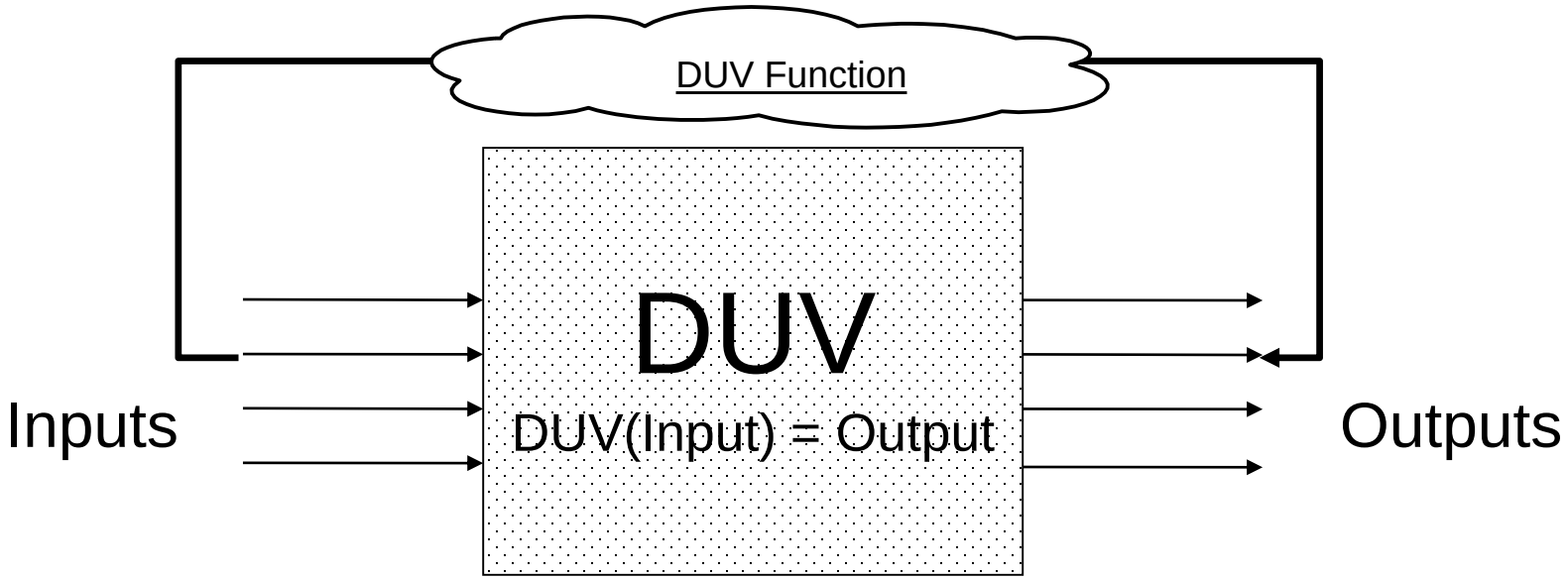
Driving the Black Box

- We can start planning the stimuli even before the complete specification of the DUV is given
- The definition of the inputs can provide information and hints on
  - The interface
  - The functionality
- This information can lead to first set of stimuli
- More stimuli will be added as we learn more details on the DUV

Checking Strategies

- In microelectronic system design there are five main ***sources of checkers***
  - The inputs and outputs of the design (specification)
  - The architecture of the design
  - The microarchitecture of the design
  - The implementation of the design
  - The context of the design (up the hierarchy)
- Note that the *source* of checkers and their *implementation* are two different issues

Checking Based On the DUV I/O



- Check the output signals of the DUV based on
  - The input signals
  - Understanding of the specification of the DUV

Checking Based On the Architecture

Example instruction stream:
SUB R7 R1 R2
BRZ R7 L

Architectural (ISA-level) checking is abundant.

- The SUB and BRZ instructions are defined in the ISA.
- Architecture defines that instructions must complete in order.
- Architecture defines that results of SUB must be used by BRZ.

Many checkers have their roots in the Architecture of the design!

Checking Based On the Architecture and Microarchitecture

- Check that architectural and microarchitectural mechanisms in the DUV are operating as expected
  - Buffers: overflow and underflow
  - Invalid states and transitions in state machines
  - Pipelines
  - Reorder buffers
  - Writeback and forwarding logic
    - performance enhancing features
  - …

Checking Based On the Implementation

- Check items that are related to specific implementation details
  - Cyclic buffers for queues
  - Pipeline buffer stages
  - …

Driving and Checking

- You need both or you get nothing!

- To find a bug:
  - Your driver must create the failing scenario, and
  - Your checker must flag the behaviour mismatch.