JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Formal Verification: First Line of Defense for Securing your Hardware from Attacks

Vedprakash Mishra, Anshul Jain, Aarti Gupta
Intel Corporation

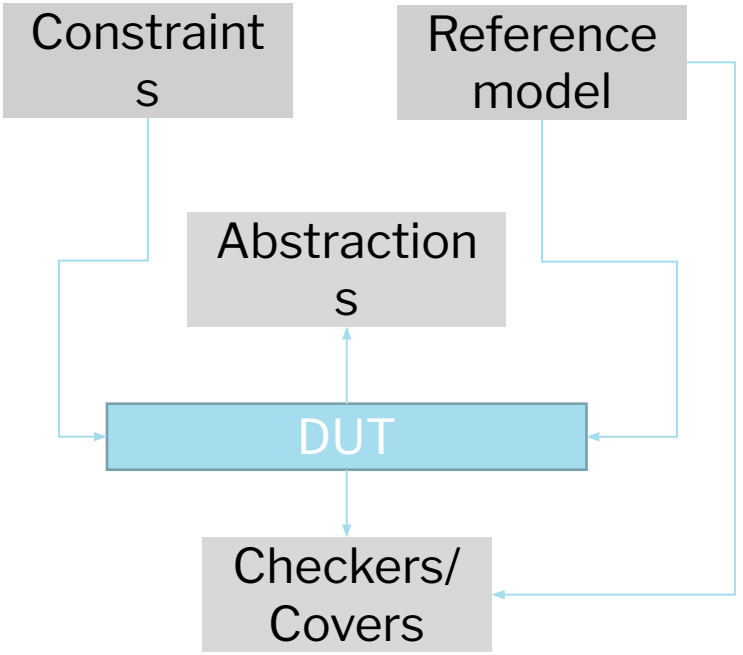
intel

Security of an IP

- In SOC, several initiators/masters such as IA cores, accelerators and devices interact with each other to process workloads and they may not be equally trustworthy
- The hardware-based attacks aim to exploit the vulnerabilities present and steal assets such as control and status registers (CSRs), keys, regions in memory, memory-mapped IO
- **The aim is not only to highlight the protection of sensitive assets but stopping security attacks from causing performance and functional breakdown**

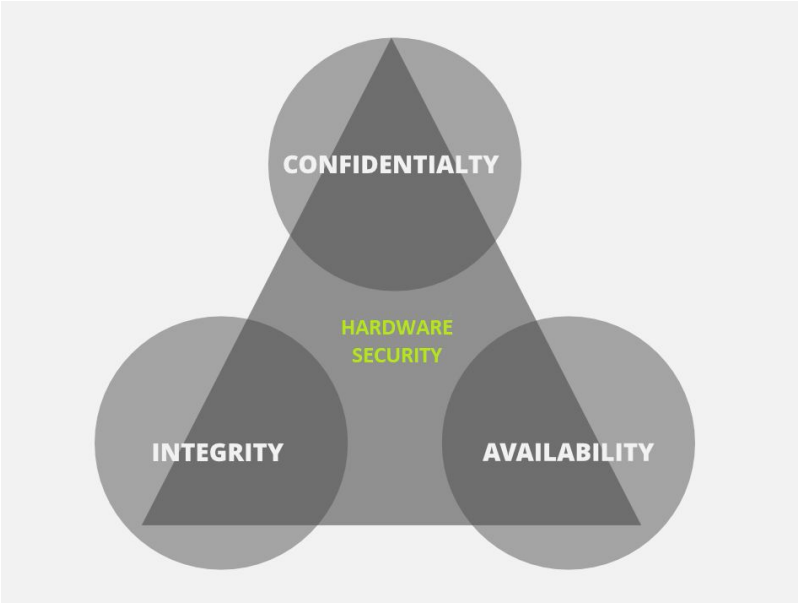
Role of FV

- Formal Verification (FV) is a mathematical technique to prove that design works as per the specifications
- **Formal Property Verification (FPV) is a powerful technique for exposing security flaws/vulnerabilities of designs "well in-time (pre-silicon)"**
- FV can be used to verify that a design satisfies a set of security properties, such as **confidentiality, integrity, and availability**
- If your IP handles secure information of the SOC chassis, then this talk will
  - Help in Security Verification Planning through Formal Verification
  - Create awareness about potential attack scenarios and their implications
  - Prepare to take risk mitigation steps and **achieve shift-left**



CIA Triad

- **Confidentiality**: Data is only accessible to authorized parties
- **Integrity**: Critical data/transaction is not tampered with or modified during transportation
- **Availability**: System is operational and accessible when needed

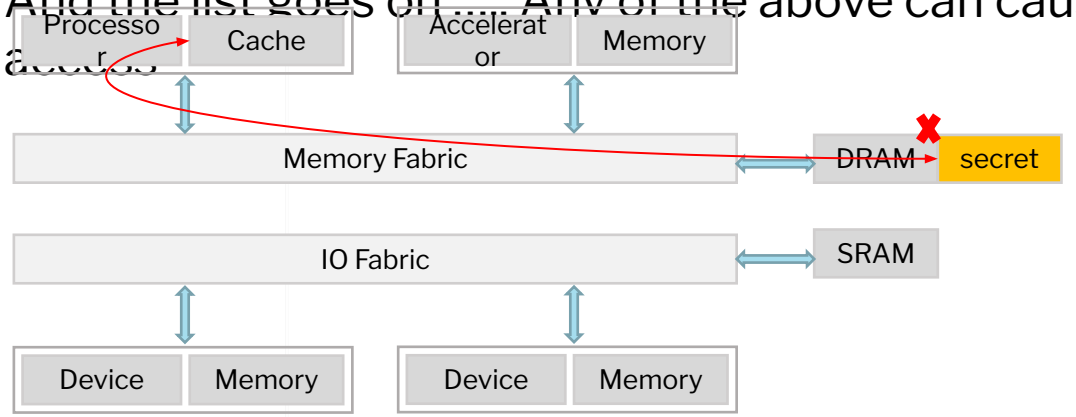


By verifying that a design satisfies these properties, formal verification can help ensure that a hardware design is secure and free from vulnerabilities

Confidentiality: Secure Access of CSRs

1. Insecure source or destination IDs
2. Critical input determining flow of transaction exposed to user
3. Un-syncing micro-architectures to affect the decoding of transaction
4. Bypassing the decoding by corrupting critical headers/bits of transaction

And the list goes on…… Any of the above can cause unsecure access



Every access initiated by an initiator/master must include attributes that must identify it or its role. Similarly, every responder/slave must include in its response attributes that identify the responder or its role

What happens if the initiator attributes are modified in the transit?

A fake or spoofed attribute may end up granting an untrusted initiator access to an asset while it should have been denied

How FPV is useful here?

FPV can be used to ensure that transactions having incorrect attributes (E.g., Src/Dest IDs) i.e., not belonging to allowed hardware allowed Port IDs should not be propagated.

```
1  wire [SRC_MSB:SRC_LSB] sym_src_port_id;
2  sym_src_port_id_constant: assume property (
3     @(posedge clk) disable iff(!rst_b)
4     ((sym_src_port_id != SOURCE_CHANNEL_1) &&
5      (sym_src_port_id != SOURCE_CHANNEL_2) &&
6       …
7       …
8      (sym_src_port_id != SOURCE_CHANNEL_N)
9     ##1 (sym_src_port_id == $past(sym_src_port_id))
10 );
```

```
1  // Outgoing transactions source flit should not match
   with illegal sym_src_port_id
2  source_info_should_be_correct: assert property (
3     @(posedge clk) disable iff(!rst_b)
4     out valid && src flit vld  |->
5     flit[SRC_MSB:SRC_LSB] != sym_src_port_id
6  );
```

Similarly, properties like “if a particular transaction is filtered (based on the attributes) then it cannot be seen on the output interface” should also be written

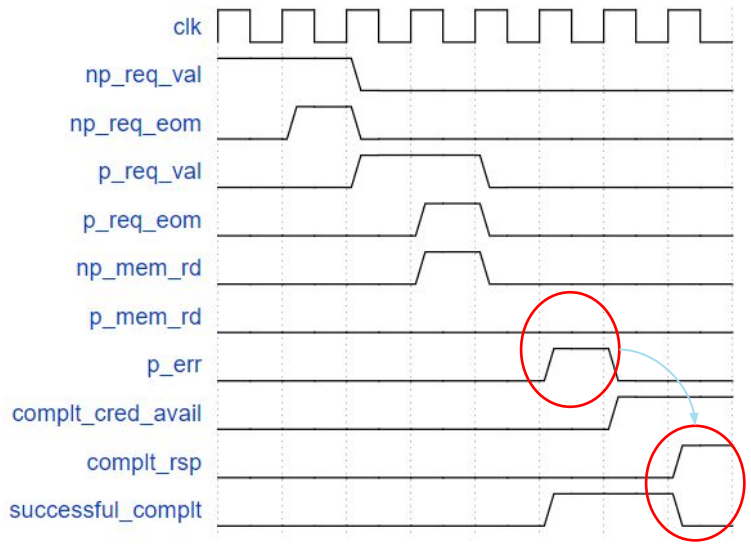
Integrity: Accurate and consistent Service

A legal non-posted transaction requires a successful/unsuccessful completion response
What if intruder's illegal transaction changed completion response of legal transaction from successful to unsuccessful or vice-versa causing **manipulation of service**???

How FPV is helpful here?

1. **Functionality correctness checkers**:
If the transaction is legal as per decoded attributes and read/write of config register is allowed/read, then only a successful completion should be sent otherwise not

2. **Data correctness checker**:
If config register was read, then the completion data sent should match the read data



The above figure illustrates the manipulation-of-service bug caught through FPV. In this case, the corrupted posted was correctly decoded and error was flagged. Completion response (complt_rsp) corresponding to non-posted request was not asserted due to unavailability of credits. As soon as credits become available, the very next cycle complt_rsp got asserted but successful field got de-asserted because of posted error (p_err)

Availability: Hang free system (No DoS)

1. Let's say, the decoding went right, and IP was able to detect that the transaction is corrupt. But did IP discard that transaction or raised an error? **(Number of flits sent by intruder is incorrect, FSM is stuck waiting for correct number of flits)**

2. Let's say IP dropped the transaction and freed the pipeline. But is it really, correct? **(Non-posted transaction requires completion response, otherwise the entire architecture will hang, legality of transaction is late-stage game)**

3. What if the corrupted transaction was a non-posted transaction; shouldn't an unsuccessful completion is required for such transaction**? (Illegal non-posted should not get a successful completion)**

4. Let's say IP sent an unsuccessful completion, but instead of sending one completion response sent two **(One request-one response)**

How FPV is useful here?

1. **Safety and Correctness** : If an illegal transaction is received then unsuccessful completion corresponding to this transaction should be sent out within finite number of cycles

2. **Spuriousness**: Number of completion responses sent should never be more than number of non-posted request received

```
reg [COUNT_W:0] count;
always @(posedge clk) begin
   if (rst) count <= 'b0;
   else if (finish) count <= 'b0;
   else if (stall && (|count))
      count <= count;
   else if (|count)
      count <= count + 1'b1;
   else if (start && !(|count))
      count <= 'b1;
end
```

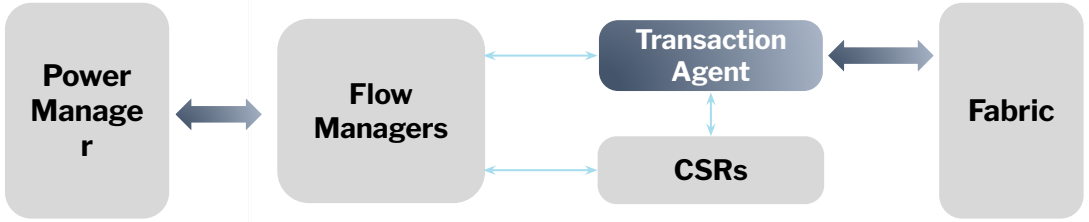
```
counter_underflow_forbidden:
assert property (
   @(posedge clk) disable iff (rst)
   count <= MAX_COUNT
);

start_condition_is_met:
cover property (
   @(posedge clk) disable iff (rst)
   start
);
```

```
reg [COUNT_W:0] count;
always @(posedge clk) begin
   if (rst) count <= 'd0;
   else begin
      count <=
         count + np_req - complt_rsp;
   end
end
```

```
counter_underflow_forbidden:
assert property (
   @(posedge clk) disable iff(rst)
   count == 'd0 |-> !complt_resp
);
```

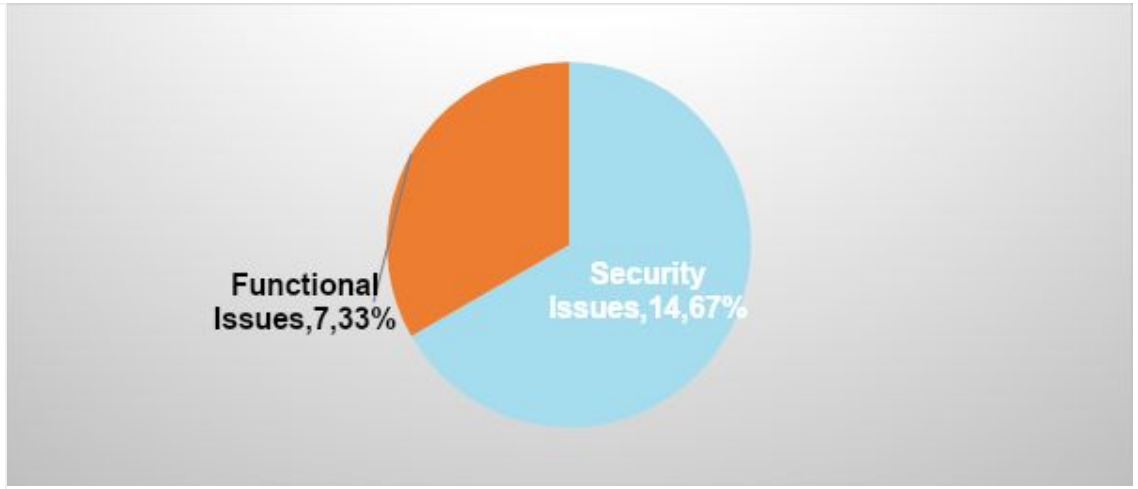
Results

Security Flaws Found With Formal Property Verification



Timelines

1. **Unsecure Access to CRs**: Transaction Agent (TA) claims transactions without security attributes, allows the attackers to modify/access Control/Status Registers (CSRs) (**2 vulnerabilities**)
2. **Denial of Service/ Hang**: TA fails to check header containing security attributes for malformed transaction, exposes TA to attackers for create hangs (**3 vulnerabilities**)
3. **Manipulation of Service**: TA unable to ignore malformed transactions appropriately, allows attackers to corrupt successful responses by injecting malformed transactions (**1**



Issues related to Integrity and availability are hard to catch by formal apps like Security Path Verification (SPV). FPV has selective advantage over here because of its exhaustiveness