2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Forward Progress in Formal Verification
Liveness vs Safety

Ankit Garg

NVIDIA®

accellera SYSTEMS INITIATIVE

Problem

- Verification is HARD
- Everything 100% verified according to spec
  - Per Spec Input ===Guarantees===> valid output
- How to guarantee
  - Wrong Input |===> Invalid output/Error
  - If system Hangs, there’s no output

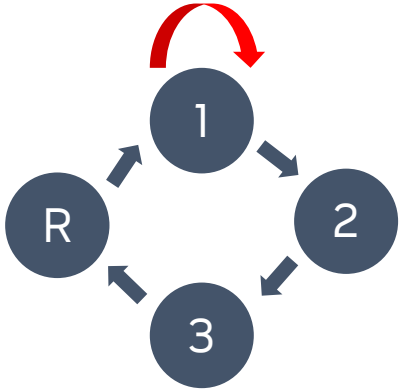
Why

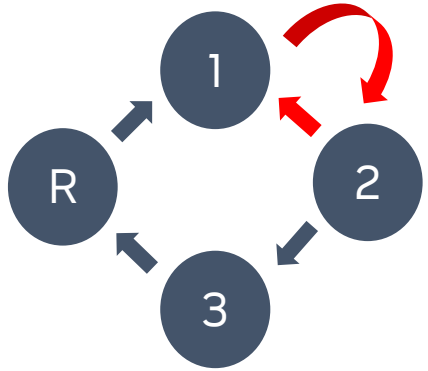
Design	Router block for high performance data center solutions
Purpose	Interact with internal/external system controllers
Problem	Can’t guarantee external system controllers stick to the spec

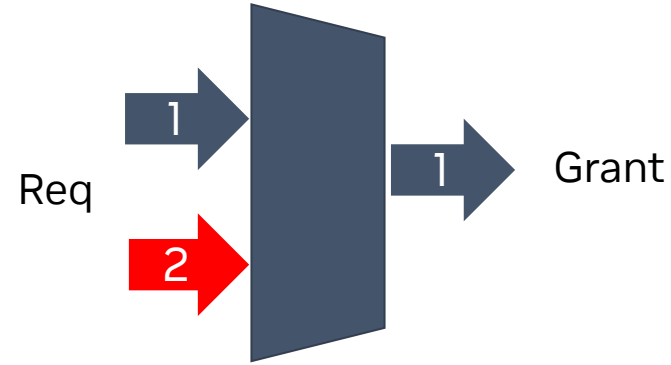
- Irrespective of what controller drives, **System should never HANG**
- Forward progress testing is critical to ensure this doesn’t happen

Forward Progress

- Making progress towards its intended goal without getting stuck.
- Means design shouldn’t have
  - Deadlock (a transaction stuck in a non-default state and can’t move)
  - Livelock (a transaction is changing states but doesn’t reach the end goal)
  - Starvation (transaction is being blocked due to low priority)







How

Simulation based Verification

- Corner Case tests
- Random tests (limited by the constraints)
- Stress tests
- Error Injection tests

Formal Verification

- Safety assertion (nothing bad happens)
- Liveness assertion (something good eventually happens)

Liveness vs Saftey

Liveness assertions

```
assert property (A |-> strong(##[0:$]B))
```

- Are simpler to write
- Wider scope
- Slower convergence
- Can’t rely on bounds
- Need aggressive abstraction
- Not enabled in simulation verif

Safety assertion

```
assert property ( A |-> ##N B )
```

- Complex to write
- Narrow scope
- Faster to converge vs Liveness
- Bounded analysis can be done
- Would need less abstraction
- Can be enabled in simulation verif

Safety assertion-based flow

- FIFO overflow and underflow check
- State Machine never enter invalid state
- Timeout counters reset check
- Transaction order check
- Protocol violation check
- Counter based forward progress checks

```
assert property (flag_bit && ~system_stall |-> ##[0:N-M] B)
```

Liveness assertion-based flow

- Identify Deadlock/Livelocks
  - Liveness assertions on cyclic logic FSMs & Counters

```
assert (State==~(reset_state) |-> strong(##[1:$] State == reset_state))
```

- Identify Starvation points
  - Liveness assertions on Arbiters, credit-valid, request-valid paths

```
assert (req == x |-> strong(##[0:$] grant==x))
assert (credit ==0 |-> strong(##[0:$] reload_credit))
```

Challenges

- Convergence
  - Biggest challenge, as even a simple 5 state FSM could have huge COI (Cone of Influence) and never fully prove
- Complexity reduction
  - Reduce the COI of assertion, without over-constraining or losing important design elements
- Getting constraints right
  - It’s a challenge to figure out minimum constraints yet have proofs
- Proof Maintenance
  - A small design change can throw a proven Liveness assertion to not converge

Techniques

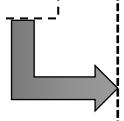
- Abstraction
  - Memories/FIFO abstraction: hit corner case faster
  - Counter abstraction: jump states, especially for slower protocols
  - Reset abstraction: hit deep states with tool's proof depth limits

- Boundary Boxing & Driver Snipping
  - Under-constraint (leave snipped port as is)
  - Over-constraint (drive constant value)
  - Correct constraint (connect assume model of the driver)

Techniques cont….

- Assume guarantee

```
assert (seq_1|-> strong(##[1:$]seq_N))
```

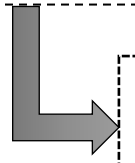


```
assume (seq_1|->strong(##[1:$] seq_2))
assume (seq_2|->strong(##[1:$] seq_4))

assert (seq_1|->strong(##[1:$] seq_N))
```

- Splitting liveness assertions (case split)

```
State==~(reset_state) |-> strong(##[1:0] state==reset_state)
```



```
state==state_1 |-> strong(##[1:$]state==reset_state)
State==state_2 |-> strong(##[1:$]state==reset_state)
...
```

Gotchas

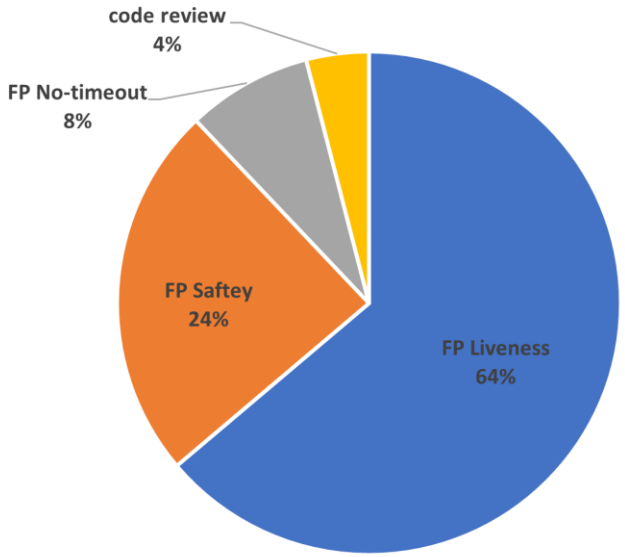
- Don’t use weak liveness assertions
- There’s no bounded proven in liveness assertions
- Setup formal regression
- Avoid mixing liveness and safety assertions in one run
- For faster convergence of a liveness assertion
  - Don’t add more resources ✗
  - Reduce complexity ✓

Case Study - Formal setups

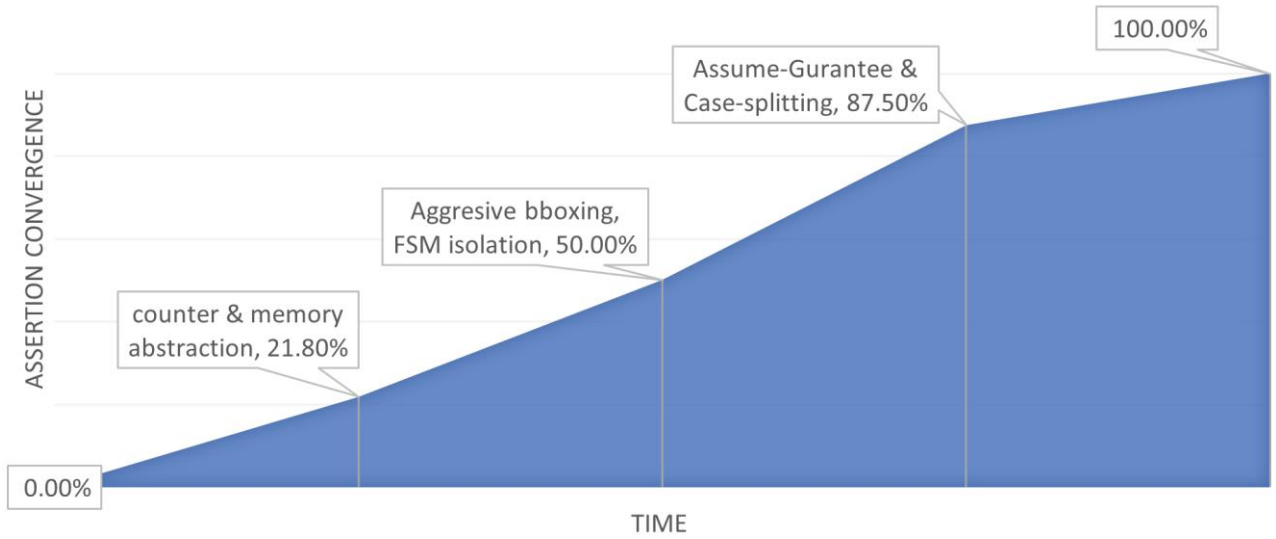
- Forward progress liveness
  - All liveness assertions
  - Minimal constraints (safety & Liveness)
- Forward progress safety
  - All safety assertions
  - All necessary constraints (safety)
- Forward progress no de-hang
  - All functionality assertions
  - All necessary constraints
  - All de-hang logic snipped

Case Study - Run Results

- Bug Distribution



- Convergence improvement



Case study - Summary

- Liveness assertions

```
assert property (A |-> strong(##[0:$]B))
```

- Faster bring up
- More bugs found
- Wider scope, better coverage
- High verification confidence

- Safety assertion

```
assert property ( A |-> ##N B )
```

- Faster convergence
- Faster to sign-off
- Coverage in simulation environment

Conclusion

- Liveness assertions while late to full proofs are fast with bugs.
- Start with Liveness assertions, while working on safety assertions.
- Safety did find some corner case failures which liveness couldn’t.
- Recommend to use healthy mix of both.

Questions ?