Efficient Bug-Hunting Techniques Using Graph-Based Stimulus Models

Nguyen Le, Microsoft Corp

Mike Andrews, Mentor Graphics Corp

Agenda

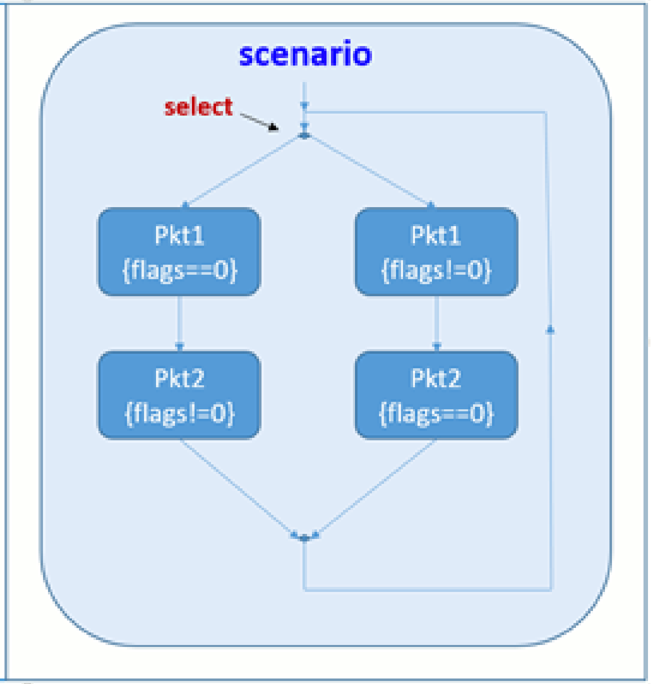
- A closer look at a graph-based model
  - Coverage goals as part of the stimulus model
  - Portable stimulus – a new standard
- Bug-hunting techniques
  - Traditional vs graph-based approaches
- Bug hunting strategies – the theory
- Setting up the experiment
- Results
- Conclusion

Graph-Based Stimulus

- Written as a declarative rule
  - Resembles Extended Backus-Naur Form (EBNF)
  - Allows “(do A) OR (do B)” - a significant advantage

```
action scenario {
    pkt        pkt1;
    pkt        pkt2;

    graph {
      repeat {
        select {
          {
            pkt1 with {flags == 0;};
            pkt2 with {flags != 0;};
          }
          {
            pkt1 with {flags != 0;};
            pkt2 with {flags == 0;};
          }
        }
      }
    }
}
```

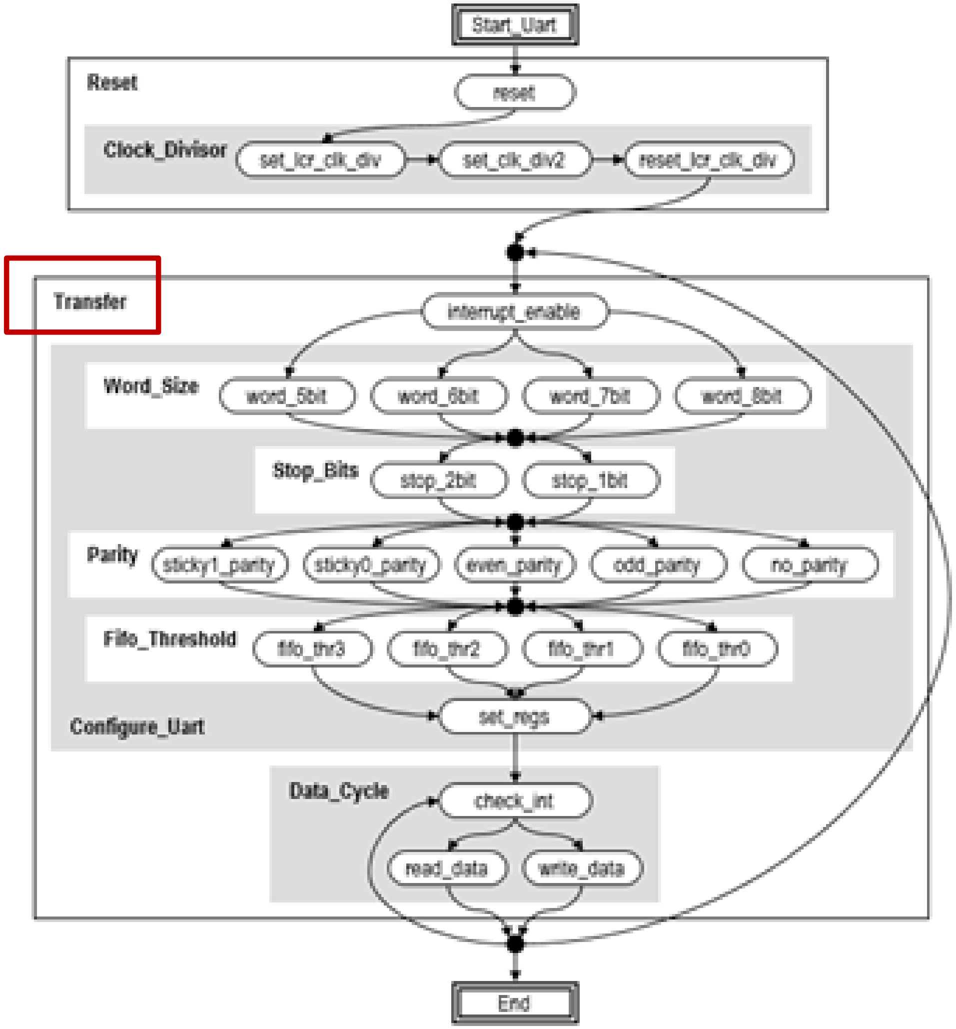


The rules define the graph, which is typically hierarchical

[Syntax borrowed from an Accellera PSWG example]

Coverage as an Input

- Common with SV:
  - Which variables?
  - Combinations?
  - Any binning?
- Unique to graphs:
  - Which paths?
  - Through which region?
  - E.g. all paths through 'Transfer' region



Portable Stimulus?

- New standard, in the early stages
- Based on contributions from graph-based methodologies
- Requirements:
  - Independent of the verification environment
  - Encompasses capabilities of current tools

"With this proposed standard, user companies will be able to specify the behaviors once, from which multiple implementations may be derived."

(Ref: Portable Stimulus Working Group. Scope Definition)

Bug-hunting Techniques

- Directed Testing:
  - Analyze the spec
  - Write tests
  - Repeat until tape-out
- Constrained Random Verification:
  - Analyze the spec
  - Write constraints
  - Write a lot of procedural code in sequences
    - To exercise specific corners of concern

Constrained Random

- High volume of random tests with legal inputs
  - Sometimes called soak testing
  - Resource intensive, low efficiency
    - But also low in human effort
- ‘Directed’ random – steered towards coverage goals
  - The typical CRV methodology
    - Coverage metrics are key to validation
  - Effort required depends on breadth & depth of goals
    - Closure generally considered a challenge

Graph-Based Techniques

- Analyze spec
- Write a graph
  - With constraints
- Write one or more scenario graphs
  - Each scenario graph replaces many sequences
- Systematic traversal
  - More efficiently covers legal scenarios
  - Reaches corners without manual intervention

Other Theoretical Strategies

- At least cover all fields
  - Use a sensible "auto bin max"
  - Limited use of SV 'randc' may help for simple cases
  - But, many bugs appear via combinations of fields
    - Hence the need for cross coverage metrics
- Pairwise testing (or n-tuple in general)
  - Cross each pair of fields (or triples etc.)
  - Depending on application, catch 90%+ of failures *
  - Dramatically reduces the number of tests

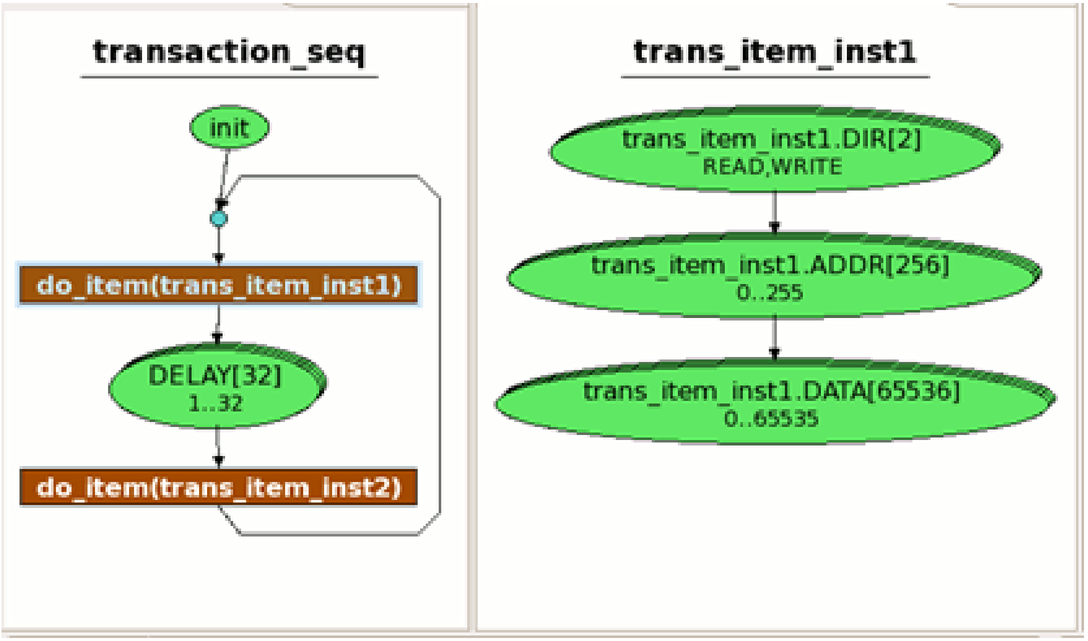
(* Ref: D. Richard Kuhn, "Practical Combinatorial Testing")

Including Sequential Effects

- Graph-based model can include multiple instances
- Declarative rule defines many possible sequences

Simple Example: 2 instances of the same transaction

Graph paths now determine transitions of transaction variables, including variable delay between them



Setting Up an Experiment

- Real design verification project
  - Secure compute block, Microsoft Xbox team
- Using code coverage as a proxy for bug-detection
  - In this design FSM transitions are a key metric
- Two sequences, 50 fields total
  - Designer insight suggests 6 fields are key
- Baseline defined by 5 parallel random runs
  - 3,371 packets generated
  - 95.9% overall coverage, 85.8% FSM Trans

Graph-based Variants

- Manually crafted strategy:
  - Cross of all 6 key fields
  - Manually crafted bins based on design knowledge
  - Total of 10,800 bins for the cross
- Automated strategies:
  - Fields test – 6 key fields, auto bin max = 258
    - Bins are one on each edge, 256 between
  - Two pairwise tests – same 6 fields
    - auto bin max of 66 & 258
  - Triples test – auto bin max = 66

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Results

- Code coverage: strategy comparisons:

Strategy Name	Maximum Bins/fields	Largest Goal	Num Packets	Total (Files)	Total (Instance)	FSM Trans
Rand	N/A	N/A	3371	95.9	97.7	85.8
BigCross	N/A	10,800	13,666	100	100	100
Fields	258	258	439	97.1	97.8	86.7
Pwise	66	660	1,026	98.9	98.1	96.4
PwiseLrg	258	2,580	4,440	100	99.8	100
Triples	66	1,320	2,682	99.7	99.8	100

Largest goal: biggest cross coverage goal in the strategy
Num Packets: Number of packets simulated to meet goal(s)

Conclusions

- Three simple take-aways
  - No substitute for designer understanding
    - 100% across the board for manually crafted cross
  - Basic automated strategies significantly outperform random
    - Fields strategy, minor improvement, 6x efficiency
      - Allows for fast but coarse validation of changes
    - Ntuple strategy, huge improvement, same resources
      - 'Most' of the benefit of manual, fraction of the effort