Formal Bug Hunting with “River Fishing” Techniques

Mark Eslinger, Mentor
Ping Yeung, Mentor
Agenda

• What is River Fishing
• Traditional approach
• River fishing approach
• Results
• Design bugs
• Summary
River Fishing

Many anglers consider river fishing to be one of the most relaxing freshwater fishing experiences because it doesn’t require much gear, and can easily be done from a canoe, kayak, or while wading. Find river fishing tips and more information.
Waypoints, Goal-posting, Cover-points

River Fishing

• Launching formal verification from interesting fishing spots
• “Selection of initial states for formal verification,” US7454324.
Formal Bug Hunting with “River Fishing”

Fishing Spots

Bug Hunting Spots

- Outside interactions
- Control and interrupts
- Concurrent events
- Feedback, Loops, Counts
Identify “fishing spots”

<table>
<thead>
<tr>
<th>Outside interactions</th>
<th>Inter-module communication and standard protocol interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control and interrupts</td>
<td>FSMs, bus controllers, memory controllers, flow charts, algorithmic controls</td>
</tr>
<tr>
<td>Concurrent events</td>
<td>Arbiters, interrupts, schedulers, switches, multiplexing logics etc</td>
</tr>
<tr>
<td>Feedback, loops, and counts</td>
<td>FIFOs, timers, counters, data transfers, bursting, and computations</td>
</tr>
<tr>
<td>Assert and cover properties</td>
<td>fan-in cones of the user properties are great coverpoints and sub-goals</td>
</tr>
</tbody>
</table>

- “Selection of initial states for formal verification,” US7454324
Identify “fishing spots”

- “Selection of initial states for formal verification,” US7454324
The 3 Major Steps

- The “river fishing” formal bug hunting methodology consists of:
Screening with Engine Health

• Formal engine health:
  – Formal targets concluded (proven/fired/covered/uncoverable)
  – Sequential depth explored or cone of influence analyzed
  – Formal knowledge and engine setting acquired
• Initially most of the 33 targets were inconclusive (I).
• With multi-cores running concurrently, formal verification gradually verified the targets into one of the following catalogs: firing (F), vacuous (V), uncoverable (U), covered (C), and proof (P).
Formal engine knowledge snapshot

<table>
<thead>
<tr>
<th>#</th>
<th># Proven / Unsatisfiable</th>
<th># Fired / Satisfied</th>
<th># Inconclusive Targets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Safety</td>
<td>Vacuity</td>
<td>Safety</td>
</tr>
<tr>
<td>0*</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7*</td>
<td>168</td>
<td>8</td>
<td>102</td>
</tr>
<tr>
<td>10*</td>
<td>29</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12*</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17*</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- The “Proven/Unsatisfiable” columns show which engines solve the safety/vacuity checks.
- The “Fired/Satisfied” columns show which engines generate the counterexamples.
- The “Inconclusive Targets” columns (Good, Fair, Poor) show the individual engine health.
- Engine 7 is very productive in finding a lot of proofs and firings.
- Engine 0 (the housekeeping engine) and Engine 10 have found some proofs.
- Engines 12 and 17 haven’t been contributing to the results.
## Results

### Block-Level Results

<table>
<thead>
<tr>
<th>Block</th>
<th>Targets</th>
<th>Fishing Spot S0</th>
<th>Fishing Spot $\sum$ SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block De</td>
<td>12</td>
<td>12 C, depth 23</td>
<td>12 C, depth 23</td>
</tr>
<tr>
<td>Block Pc</td>
<td>71</td>
<td>66 C + 5 I, depth 134</td>
<td>71 C + 0 I, depth 256</td>
</tr>
<tr>
<td>Block Cp</td>
<td>81</td>
<td>60 C + 21 I, depth 65</td>
<td>79 C + 2 I, depth 161</td>
</tr>
</tbody>
</table>

C: proven/fired/covered/uncoverable. I: inconclusive targets
### Results

#### Block-Level Results

<table>
<thead>
<tr>
<th>Block</th>
<th>Targets</th>
<th>Fishing Spot S0</th>
<th>Fishing Spot ∑ SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block De</td>
<td>12</td>
<td>12 PF, depth 23</td>
<td>12 PF, depth 23</td>
</tr>
<tr>
<td>Block Pc</td>
<td>71</td>
<td>66 PF/CU + 5 I, depth 134</td>
<td>71 PF/CU + 0 I, depth 256</td>
</tr>
<tr>
<td>Block Cp</td>
<td>81</td>
<td>60 PF/CU + 21 I, depth 65</td>
<td>79 PF/CU + 2 I, depth 161</td>
</tr>
</tbody>
</table>

#### Sub-System Level Results

<table>
<thead>
<tr>
<th>Design</th>
<th>Targets</th>
<th>First stage</th>
<th>Fishing spot S0</th>
<th>Fishing spot ∑ SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Ct</td>
<td>2356</td>
<td>2319 PF/CU + 37 I 15 min</td>
<td>2338 PF/CU + 18 I 24 hours</td>
<td>2349 PF/CU + 7 I 24 hours</td>
</tr>
<tr>
<td>Design Pb</td>
<td>15205</td>
<td>15126 PF/CU + 79 I 15 min</td>
<td>15154 PF/CU + 51 I 24 hours</td>
<td>15161 PF/CU + 44 I 24 hours</td>
</tr>
</tbody>
</table>

PF/CU: proven/fired/covered/uncoverable. I: inconclusive targets
Case #1: Design Interface Bugs

- **Ratio-synchronized Data Sampling**
  - Sub-systems are running at slower frequencies

- **Assertions**
  - Ensure the data is sampled at the right time

- **Bug**
  - The fast clock domain samples the data at the end of a slow clock period
  - In some corner cases, Data Valid condition is not checked
  - As a result, the data is sampled at the incorrect time, and corrupted data is registered
Case #2: Data Transfer Controller

- **DMA Controller**
  - DMA Controller in WLAN/PDA design
- **Assertions**
  - Monitors for on-chip bus interfaces
  - Pointer manipulation, allocation and de-allocation
- **Bug**
  - Channels are set up to handle data with priority
  - When >1 channels finish the transfer at the same time
  - One address pointer is de-allocated twice, while the other is not de-allocated
  - Causes memory leak and data corruption
Summary

- Simulation and formal methodologies can be used together to accelerate the verification of complex designs.
  - Leverage what has been learned or achieved in one for the other
  - Some companies have already made organizational changes

- *River fishing* technique
  - Leverages the functional simulation activities and starts formal verification from interesting fishing spots in the simulation traces.
  - Identify and extract a set of good fishing spots from the simulation traces
  - Screen and prioritize the fishing spots using formal engine health
  - Launch and monitor multiple formal runs on the computing servers
Thank you