24. New Directions in Verification

Jacob Abraham
Department of Electrical and Computer Engineering
The University of Texas at Austin
Verification of Digital Systems
Spring 2020
April 23, 2020

Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology
Verification is a Fundamental Technology

- Dealing with the analysis of extremely complex systems
- Can answer questions about the behavior of systems
- Verification algorithms and abstraction techniques can be applied to a variety of application problems
  - Generating at-speed (functional/application-level) tests for faults in an embedded module in a VLSI chip
  - Identifying accurate application-level power consumption from module-level power information
  - Automatically identifying portions of a chip which can be gated off during a particular clock cycle (to reduce power)
  - Automatically finding “implications” of a given pattern in a data set
  - Improving statistical correlations in data mining
- The verification problem will never go away, as long as there are designs
  - Will always need to verify the correctness of designs

Look at the Big Picture

Verification can also be applied to the other areas involving not only integrated circuits, but also any problem which can be modeled using logic functions and where answers to logic questions are desired.
Applicability of Verification Techniques

### Simulation-Based Verification
- Can simulate very large designs
- Drawback: only a very small fraction of all possible inputs can be simulated in practice
- Generally use random or targeted sequences to achieve a coverage goal

### Formal Verification
- Can only deal with small blocks
- Definite answer to whether a property holds, or a counter-example trace (assume the block is within the capacity of the tool)
- Useful for checking uncovered states, properties, etc., to achieve a coverage target

Application of Machine Learning

- Relatively new area of research
  - Lot of hype
  - Techniques “learn” from data generated from the system, UVM tests, for example
  - Genetic algorithms (discussed previously) are a type of “reinforcement learning”

Some questions to ponder when considering using machine learning for verification

- Can we use learning to find a bug, or prove the absence of a bug?
- If massive amounts of data are needed for high quality learning, are there other techniques which would require less effort?
- Can the data point to areas not covered by the tests?
- Can additional information regarding the behavior of the system be deduced (learned)?
Outline

- Introduction
- **Testing circuits after manufacture and in the field**
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Software-Based (Native-Mode) Self Test for Processors

- Why not use functional capabilities of processors to replace BIST hardware?
  - No additional hardware
- Reduce test costs by using low-cost testers
- Increase coverage of delay defects and increase yield by testing native
- No issues with excessive power consumption during test

**Developed at University of Texas (Int’l Test Conference 1998)**

**Application to processors at Intel (Int’l Test Conference 2002)**
Are Random Tests Sufficient?

Intel implementation involved code in the cache which generated random instruction sequences
Interest in generating instructions targeting faults
Possible to generate instruction sequences which will test for an internal stuck-at fault in a module
In order to deal with defects in DSM technologies, need to target small delay defects
Recent work: automatically generate instruction sequences which will target small delay defects in an internal module

Tests for Small Delay Defects

Need to test paths in the circuit to detect small delay defects
However, the number of paths in a circuit can be exponential in the number of nodes
Solution: test the longest path through every node
  
  This will detect the smallest possible delay increase which will cause the circuit to fail
Total number of tests is linear in the number of nodes
Automatic Generation of Instruction Sequences for Small Delay Defects

- Phase 1: all paths above a delay threshold
- Phase 2: longest paths through all nodes
- Delay-Based ATPG: generate "TRUE" paths above given delay threshold
- **Functional mapping:** using verification engine

**Verification Property to Generate Test Sequence**

<table>
<thead>
<tr>
<th>Path</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net</td>
<td>Transition</td>
</tr>
<tr>
<td>InsA.Y</td>
<td>Rising</td>
</tr>
<tr>
<td>InsB.Y</td>
<td>Falling</td>
</tr>
<tr>
<td>InsC.Y</td>
<td>Rising</td>
</tr>
</tbody>
</table>

Modified property with instructional constraints

```c
always
if (`insn_legal && (InsA.Y == 0) && (InsB.Y == 1) && (InsC.Y == 0)) begin
  wait(1);
if (`insn_legal && (InsA.Y == 1) && (InsB.Y == 0) && (InsC.Y == 1)) begin
  wait(1);
......
```
Results on OR1200 processor

www.opencores.org, synthesized for 0.18µ TSMC process

Results for Phase 1 (paths > 80% of clock)

<table>
<thead>
<tr>
<th>No. of Paths</th>
<th>Drop</th>
<th>Functionally Testable</th>
<th>Functionally Redundant</th>
<th>Time out</th>
</tr>
</thead>
<tbody>
<tr>
<td>27424</td>
<td>12</td>
<td>15118</td>
<td>12106</td>
<td>200</td>
</tr>
</tbody>
</table>

Results for Phase 2

N: % nodes with test for longest path through them

<table>
<thead>
<tr>
<th>Module</th>
<th>Functionally Testable</th>
<th>Functionally Redundant</th>
<th>Rejected Sub-paths</th>
<th>N (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>or1200_ctrl</td>
<td>1826</td>
<td>29191</td>
<td>68087</td>
<td>90.6</td>
</tr>
<tr>
<td>or1200_alu</td>
<td>1427</td>
<td>16985</td>
<td>2716</td>
<td>100</td>
</tr>
<tr>
<td>or1200_lsu</td>
<td>970</td>
<td>4077</td>
<td>3744</td>
<td>100</td>
</tr>
<tr>
<td>or1200_wbmux</td>
<td>1146</td>
<td>2285</td>
<td>2118</td>
<td>100</td>
</tr>
</tbody>
</table>

Test of SoC Cores using Embedded Processor (Gurumurthu et al., 2008)

Wishbone and 128-bit AES designs from opencores.org
Validation vectors: random values encrypted/decrypted

<table>
<thead>
<tr>
<th>AES Core</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Combinational primitives</th>
<th>Sequential primitives</th>
<th>Stuck-at faults</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>69</td>
<td>33</td>
<td>9225</td>
<td>1119</td>
<td>64070</td>
</tr>
</tbody>
</table>

Result of Mapping AES tests to ARM instructions (one case)

<table>
<thead>
<tr>
<th>Test</th>
<th>Size (bytes)</th>
<th>Fault coverage(%)</th>
<th>Original Coverage(%)</th>
<th>No. of Cycles</th>
<th>Original Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9128</td>
<td>90.15</td>
<td>90.35</td>
<td>7816</td>
<td>7435</td>
</tr>
</tbody>
</table>
RT Level Test Generation Targeting Hard-to Detect Faults
(Prabhu et al., 2012)

Overview
- Map gate level stuck-at fault to RTL
- Capture the propagation constraints as an LTL property
- Generate a witness for the LTL property using bounded model checking
- Use SMT-based bounded model checking
- Scale with cone of influence reduction

Modeling Stuck-at Faults in RTL

Approach
- Assume one-to-one match between flops in RTL, netlist
- Identify flops/primary outputs $o_1, o_2, ... , o_n$ in output cone of the fault
- Identify the boolean function for each of the output flops/primary outputs. Ex $o_k = f_k(i_1, i_2, ... , i_m)$
- Identify the boolean function for the output flops with the fault inserted. Ex $o_k^f = f_k^f(i_1, i_2, ... , i_m)$
- Fault condition: $fault_k = f_k(i_1, i_2, ... , i_m) \oplus f_k^f(i_1, i_2, ... , i_m)$
Example

- `always @(posedge clk) sum <= PI ⊕ sum;`
- `sum = (PI ∧ ¬sum) ∨ (¬PI ∧ sum)`
- `sum_f = PI ∧ ¬sum`
- `fault_sum = sum ⊕ sum_f = ¬PI ∧ sum`
### Experimental Setup

**Process**
- OR1200 RISC processor was DUT (included multiplier in data path)
- EBMC Model checker / Boolector SMT solver
- Bound of pipeline depth + 1
- Focused on hard to detect faults in control logic
- Commercial ATPG to sieve out easy to detect stuck-at faults
- 78% Fault coverage with commercial ATPG

### Experimental Results

<table>
<thead>
<tr>
<th>Module</th>
<th>ATPG FC(%)</th>
<th>Flts.</th>
<th>SAT based method FC(%)</th>
<th># TO</th>
<th>T(sec)</th>
<th>Naive Observability Method FC(%)</th>
<th># TO</th>
<th>T(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>80.35</td>
<td>328</td>
<td>84.11</td>
<td>310</td>
<td>96.18</td>
<td>88.49</td>
<td>161</td>
<td>95.13</td>
</tr>
<tr>
<td>ctrl</td>
<td>63.21</td>
<td>832</td>
<td>65.97</td>
<td>817</td>
<td>83.12</td>
<td>87.15</td>
<td>59</td>
<td>69.72</td>
</tr>
<tr>
<td>operands</td>
<td>75.56</td>
<td>378</td>
<td>75.59</td>
<td>354</td>
<td>95.49</td>
<td>98.36</td>
<td>6</td>
<td>57.46</td>
</tr>
<tr>
<td>srs</td>
<td>89.59</td>
<td>393</td>
<td>90.85</td>
<td>381</td>
<td>93.71</td>
<td>93.78</td>
<td>57</td>
<td>90.27</td>
</tr>
<tr>
<td>freeze</td>
<td>82.94</td>
<td>87</td>
<td>96.14</td>
<td>2</td>
<td>84.91</td>
<td>100</td>
<td>0</td>
<td>43.51</td>
</tr>
<tr>
<td>if</td>
<td>76.59</td>
<td>744</td>
<td>80.50</td>
<td>7368</td>
<td>97.57</td>
<td>90.21</td>
<td>463</td>
<td>69.33</td>
</tr>
<tr>
<td>except</td>
<td>72.69</td>
<td>1263</td>
<td>73.48</td>
<td>1209</td>
<td>98.63</td>
<td>92.79</td>
<td>128</td>
<td>96.19</td>
</tr>
<tr>
<td>Overall</td>
<td>78.05</td>
<td>10055</td>
<td>72.17</td>
<td>10343</td>
<td>96.23</td>
<td>93.86</td>
<td>874</td>
<td>76.11</td>
</tr>
</tbody>
</table>

- **FC(%)**: Fault Coverage in %
- **# Faults**: # of Undetected Collapsed Faults
- **# TO**: # of Timed Out faults
- **T(sec)**: Average Time for generating a test for a fault in seconds
Experimental Results, Structural Observability

<table>
<thead>
<tr>
<th>Module</th>
<th>FC(%)</th>
<th># TO</th>
<th>T(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>98.17</td>
<td>25</td>
<td>23.14</td>
</tr>
<tr>
<td>ctrl</td>
<td>99.21</td>
<td>8</td>
<td>21.16</td>
</tr>
<tr>
<td>oprmuxes</td>
<td>100</td>
<td>0</td>
<td>19.33</td>
</tr>
<tr>
<td>sprs</td>
<td>97.53</td>
<td>12</td>
<td>18.39</td>
</tr>
<tr>
<td>freeze</td>
<td>100</td>
<td>0</td>
<td>10.48</td>
</tr>
<tr>
<td>rf</td>
<td>98.37</td>
<td>172</td>
<td>22.85</td>
</tr>
<tr>
<td>except</td>
<td>97.63</td>
<td>69</td>
<td>38.14</td>
</tr>
<tr>
<td>Overall</td>
<td>98.87</td>
<td>454</td>
<td>24.23</td>
</tr>
</tbody>
</table>

**FC(%)**: Fault Coverage in %

**# Faults**: # of Undetected Collapsed Faults

**# TO**: # of Timed Out faults

**T(sec)**: Average Time for generating a test for a fault in seconds

Summary of Results

- Functional fault coverage of 99% for OR1200 processor
- SMT based approach was 4x faster than SAT

Coverage and Run Time Comparisons *(Prabhu et al., 2012)*
Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Early Power Estimation (RTL and Above)

Activity factor estimation

- Logic functions do not change due to synthesis, only their implementations change
- Approximate the activity at the RT-Level
  - Get input-output activity by RT-Level simulation
  - Empirical observation to obtain activity in intermediate stages,
    \[ s_{f_1} = (s_{f_{in}} - s_{f_{out}}) \times (1 - \frac{1}{N})^2 + s_{f_{out}} \]
  - Quadratic variation with respect to logic depth

Logical effort, modified to extract capacitance for any delay target

- Stage effort from delay \( f = F = \frac{D}{N} \)
- Sizing of nodes \( C_{in} = C_{out} \times \frac{N}{f} \)
Prototype Tool

RTL Verilog → Propagate Capacitance estimator
                  → Activity factor estimator
                  → Simulation
                  → Number of stages (N) → Capacitance estimator
                  → Power estimator at various delay points

Parser: RTL to CDFG

Experiments

Estimated values vs. reference values
Reference values obtained at gate-level
Interconnect: wire-load model
Libraries:
- Artisan TSMC 0.18µm
- Virtual Silicon UMC 0.13µm
Library sets: (x1, x2, x4) (2ip, 3ip, 4ip)
Circuits
- OR1200 and FPU (opencores)
- ISCAS high-level models
- ISCAS sequential circuits
Results (robust with respect to technologies and libraries)

**Combinational circuits (0.18µm)**

<table>
<thead>
<tr>
<th>Target gate library</th>
<th>Average abs. error</th>
<th>Average rel. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2 ip</td>
<td>17.12</td>
<td>8.05</td>
</tr>
<tr>
<td>1,2,3 ip</td>
<td>18.95</td>
<td>11.81</td>
</tr>
<tr>
<td>1,2,3,4 ip</td>
<td>19.60</td>
<td>17.65</td>
</tr>
</tbody>
</table>

After accuracy improvement:

**Relative error estimates for sequential circuits**

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Target gate library</th>
<th>Err% 0.13</th>
<th>Err% 0.18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral</td>
<td>1,2 ip</td>
<td>6.48</td>
<td>6.80</td>
</tr>
<tr>
<td></td>
<td>1,2,3 ip</td>
<td>5.75</td>
<td>7.89</td>
</tr>
<tr>
<td></td>
<td>1,2,3,4 ip</td>
<td>5.47</td>
<td>6.79</td>
</tr>
<tr>
<td>Structural ISCAS</td>
<td>1,2 ip</td>
<td>8.26</td>
<td>10.19</td>
</tr>
</tbody>
</table>

Peak Power Estimation

**Objective**

Finding an instruction stream which maximizes the dynamic power, given the gate-level description of the processor.
Algorithm

0. Start

1. Read netlist, capacitance

2. Random starting config.

3. Apply local search, note best config.

4. Adjust TS parameter

5. Apply Tabu search (TS), note best config.

6. Enough iterations?

7. Enough random starts?

8. Output best config.

9. Stop

Power: Module-Level versus Processor-Level

Power in mW

Local level

Instruction stream

Department of Electrical and Computer Engineering, The University of Texas at Austin
J. A. Abraham, April 23, 2020
Automatic Annotation of RTL Code for Low Power

Instruction-driven slice of a microprocessor design

- All the relevant circuitry of the design required to completely execute a specific instruction
- Parts of the decode, execute, writeback etc. blocks
- Cone of influence of the semantics of the instruction

- Given a microprocessor design and an instruction
  - Identify the instruction-driven slice
  - Shut off the rest of the circuitry
- This might include
  - Gating out parts of different blocks
  - Gating out floating point units during integer ALU execution
  - Turning off certain FSMs in different control blocks since exact constraints on their inputs are available due to instruction-driven slicing

Approach

- vRTL
- vRTL Slicer
- aRTL
- Architectural Model
- SPEC2000 Benchmarks
- Power Model (Process parameters)
- Power Estimation Results (RTL)
- SimpleScalar (architectural simulator) with Watch
- Power Estimation Results (Architecture)
Results on a Simple Pipelined Processor (OR1200)

- Single instruction issue pipelined RISC microprocessor
  - Results shown after inserting annotations
  - Sliced on 1, 4, 10 instructions
  - For SPECINT2000 benchmarks

Similar results on PUMA (dual-issue, out-of-order super-scalar, fixed-point PowerPC core)

Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology
Longer Term Technologies

The basic principles of design do not change – just using different building blocks
Verification solutions are still relevant

Design Bugs

Logic bugs
- Verification is dominating the design cycle
- Unlikely that all design bugs are caught before deployment
- Diversity is necessary to deal with design bugs

Design margins
- Effects of real bugs are not easy to duplicate (in many cases, error latencies of many millions (or billions) of cycles)
- Gray: concepts of Bohr bugs (repeatable) versus Heisenbugs (not seen to be repeatable)

Bugs and design margins could be exploited by an attacker
Security Attacks

Hardware Trojans
- Malicious modification of designs
- Example of analog circuitry modifying a digital chip – extremely difficult to identify
- Design diversity may be a solution

External attacks
- Classic work (Abadi) suggested control flow checking to detect execution of undesired code
- Effects of attacks could include modification of data, execution sequences, denial of service, etc.
  - Require data checks in addition to control-flow checks
  - Need to detect DoS attacks during operation – example, shutting down GPS system (or spoofing GPS position)

Detecting and Preventing Intrusions

- Source: PurpleSec
Verifying Hardware Security Properties

Source: Tortuga Logic
Source: Fadiheh et. al, DATE 2019

Framework for Hardware Control Flow Monitoring
(Chaudhari et al., 2012)
Signature Computation

Control Flow Co-Processor Architecture

Department of Electrical and Computer Engineering, The University of Texas at Austin
J. A. Abraham, April 23, 2020
Verifying Safety of a Flight-Critical System

Simulink “Transport Class Model” (TCM) (from NASA) of twin-engine aircraft

- Properties derived from higher-level safety requirements


Summary of Verified TCM Properties

<table>
<thead>
<tr>
<th>#</th>
<th>Property</th>
<th>Assumptions</th>
<th>Original Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G-250</td>
<td>G-260</td>
<td>The heading control mode, when selected, sends roll commands to turn to and maintain the commanded heading.</td>
</tr>
<tr>
<td>2</td>
<td>G-110</td>
<td>G-220,G-260</td>
<td>The guidance system shall be capable of steering to and following a specified heading.</td>
</tr>
<tr>
<td>3</td>
<td>G-120</td>
<td>G-180,A1,A2,FPA1</td>
<td>The guidance shall be capable of climbing at a defined rate, to be limited by minimum and maximum engine performance and airspeed.</td>
</tr>
<tr>
<td>4</td>
<td>G-130</td>
<td>G-180,A1,A2</td>
<td>The guidance shall be capable of descending at a defined rate, to be limited by minimum and maximum engine performance airspeed.</td>
</tr>
<tr>
<td>5</td>
<td>G-140</td>
<td>G-120,G-200</td>
<td>The guidance shall be capable of climbing at a specified rate to a specified altitude, to be limited by maximum engine performance for a set airspeed.</td>
</tr>
<tr>
<td>6</td>
<td>G-150</td>
<td>G-180,A1,G-120,A2,G-200</td>
<td>The guidance shall be capable of descending at a specified rate to a specified altitude, to be limited by maximum engine performance for a set airspeed.</td>
</tr>
<tr>
<td>7</td>
<td>G-170 (Mode)</td>
<td>–</td>
<td>The altitude control shall engage when the altitude control mode is selected and when the FPA control mode is not selected, and when there is no manual pitch or manual roll command from the stick.</td>
</tr>
<tr>
<td>8</td>
<td>G-180 (Mode)</td>
<td>–</td>
<td>The FPA control shall engage when the FPA mode is selected, and when there is no manual pitch or manual roll command from the stick.</td>
</tr>
<tr>
<td>9</td>
<td>G-100</td>
<td>–</td>
<td>The Guidance system shall be capable of maintaining a steady speed in the normal flight envelope.</td>
</tr>
<tr>
<td>10</td>
<td>G-200</td>
<td>–</td>
<td>If the altitude control is engaged, once the plane is within 250 ft of the commanded altitude, the plane will remain within 250 ft of the commanded altitude.</td>
</tr>
</tbody>
</table>
Modeling Errors Affecting Verification Results

- Some components produced output when disabled (e.g., the altitude controller)
  - TCM model provided was incomplete
- Manual inputs from the pilot did not override the outputs of the autopilot for all three axes
  - Incompleteness in the TCM model
- Some inputs were not variables but appeared as fixed constant values in the model (e.g., the bank angle limit of G-240)
  - Modeling error
- Conflict of G-180 with G-210 and the implicit assumption that only the flight path angle control or the altitude control can be active at any moment in time
  - G-180 had to be refined
Applications of Verification Technology in Other Domains

- Analysis of complex systems
  - Analyzing the power grid for “green” power
  - Analyzing DNA sequences
- Analysis of emerging systems
  - Micromechanical systems
  - Microfluidics systems

Finding “Implications” in DNA Sequences

Six different types of Boolean relationships between pairs of genes taken from the Affymetrix U133 Plus 2.0 human dataset
Found using Boolean verification tools The two axes correspond to the expression levels of two genes

Source: Sahoo et al., Genome Biology 2008
Application to Systems Biology – Example Tools

- **BAM**, LDL degradation pathway
- **BIOCHAM**, Mammalian cell cycle control, G protein-coupled receptor kinases
- **BoolNet**, Genetic networks
- **COPASI**, Biochemical networks
- **GreatSPN**, Signal transduction pathways for angiogenesis
- **IBM Rational Rhapsody**, T-cell activation with statecharts
- **PRISM**, Biological signaling pathways, bone pathologies
- **Simulink**, Heart model for pacemaker verification
- **S-TaLiRo**, Modeling insulin-glucose regulatory system


Formalization and Verification of Medical Guidelines

Baumler, LNCS 3925, 2006
Probabilistic Model Checking of Complex Biological Pathways

Fibroblast Growth Factor (FGF) pathways, modeled in PRISM

Heath, Computational methods in system biology, 2006
Partial Reaction Rules for Pathway

1. FGF binds to FGFR
   \[ \text{FGF} + \text{FGFR} \rightarrow \text{FGFR:FGF} \quad (k_{a}=5 \times 10^{-8} \text{M}^{-1} \text{s}^{-1}, \; k_{d}=1 \times 10^{-1} \text{s}^{-1}) \]

2. Whilst FGFR:FGF exists
   - \[ \text{FGFR:Y553} \rightarrow \text{FGFR:Y553P} \quad (k_{a}=0.1 \text{s}^{-1}) \]
   - \[ \text{FGFR:Y654} \rightarrow \text{FGFR:Y654P} \quad (k_{a}=1.4 \text{s}^{-1}) \]

3. When FGFR Y653P and FGFR Y654P exist
   - \[ \text{FGFR:Y463} \rightarrow \text{FGFR:Y463P} \quad (k_{a}=7 \text{s}^{-1}) \]
   - \[ \text{FGFR:Y583} \rightarrow \text{FGFR:Y583P} \quad (k_{a}=7 \text{s}^{-1}) \]
   - \[ \text{FGFR:Y585} \rightarrow \text{FGFR:Y585P} \quad (k_{a}=7 \text{s}^{-1}) \]
   - \[ \text{FGFR:Y766} \rightarrow \text{FGFR:Y766P} \quad (k_{a}=7 \text{s}^{-1}) \]

4. FGFR binds FRS2
   - \[ \text{FGFR:Y444F} \rightarrow \text{FGFR:FRS2} \quad (k_{a}=2 \times 10^{-5} \text{M}^{-1} \text{s}^{-1}, \; k_{d}=2 \times 10^{-5} \text{s}^{-1}) \]

5. When FGFR Y653P, FGFR Y654P and FRS2 exist
   - \[ \text{FRS2:Y106} \rightarrow \text{FRS2:Y106P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y290} \rightarrow \text{FRS2:Y290P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y306} \rightarrow \text{FRS2:Y306P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y382} \rightarrow \text{FRS2:Y382P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y382} \rightarrow \text{FRS2:Y382P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y436} \rightarrow \text{FRS2:Y436P} \quad (k_{a}=0.2 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y471} \rightarrow \text{FRS2:Y471P} \quad (k_{a}=0.2 \text{s}^{-1}) \]

6. Reverse when Shp2 bound to FRS2
   - \[ \text{FRS2:Y106P} \rightarrow \text{FRS2:Y106} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y290P} \rightarrow \text{FRS2:Y290} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y306P} \rightarrow \text{FRS2:Y306} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y382P} \rightarrow \text{FRS2:Y382} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y436P} \rightarrow \text{FRS2:Y436} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y471P} \rightarrow \text{FRS2:Y471} \quad (k_{a}=12 \text{s}^{-1}) \]
   - \[ \text{FRS2:Y472P} \rightarrow \text{FRS2:Y472} \quad (k_{a}=12 \text{s}^{-1}) \]

7. FRS2 effectors bind phospho-FRS2:
   - \[ \text{Src+FRS2:Y106P} \rightarrow \text{Src+FRS2:Y106P} \quad (k_{a}=2 \times 10^{-5} \text{M}^{-1} \text{s}^{-1}, \; k_{d}=2 \times 10^{-5} \text{s}^{-1}) \]
   - \[ \text{Grb2+FRS2:Y382P} \rightarrow \text{Grb2+FRS2:Y382P} \quad (k_{a}=2 \times 10^{-5} \text{M}^{-1} \text{s}^{-1}, \; k_{d}=2 \times 10^{-5} \text{s}^{-1}) \]
   - \[ \text{Shp2+FRS2:Y471P} \rightarrow \text{Shp2+FRS2:Y471P} \quad (k_{a}=2 \times 10^{-5} \text{M}^{-1} \text{s}^{-1}, \; k_{d}=2 \times 10^{-5} \text{s}^{-1}) \]

8. When Src+FRS2 we relocate/remove Src+FRS2
   - \[ \text{Src+FRS2} \rightarrow \text{relocate out} \quad (t_{1/2}=15 \text{min}) \]

Transient Numerical Results

(a) Probability bound (Grb2-FRS2)

(b) Expected bindings (Grb2-FRS2)

(c) Expected time bound (Grb2-FRS2)

(d) Probability relocated (Src-FRS2)