Discrete Transition System Model and Verification for Mitochondrially Mediated Apoptotic Signaling Pathways

Huy H. Lam

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
In
Computer Engineering

Dr. Michael S. Hsiao, Chair
Dr. David C. Samuels
Dr. A. Lynn Abbott

June 20, 2007
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia

Keywords: Apoptosis, Reachability Analysis, Generic Algorithm, Fault Analysis, Model Checking, SAT, Mitochondria

Copyright © 2007, Huy H. Lam

Discrete Transition System Model and Verification for Mitochondrially Mediated Apoptotic Signaling Pathways

Huy H. Lam

Abstract

Computational biology and bioinformatics for apoptosis have been gaining much momentum due to the advances in computational sciences. Both fields use extensive computational techniques and modeling to mimic real world behaviors. One problem of particular interest is on the study of reachability, in which the goal is to determine if a target state or protein concentration in the model is realizable for a signaling pathway. Another interesting problem is to examine faulty pathways and how a fault can make a previously unrealizable state possible, or vice versa. Such analysis can be extremely valuable to the understanding of apoptosis. However, these analyses can be costly or even impractical for some approaches, since they must simulate every aspect of the model.

Our approach introduces an abstracted model to represent a portion of the apoptosis signaling pathways as a finite state machine. This abstraction allows us to apply hardware testing and verification techniques and also study the behaviors of the system without full simulation. We proposed a framework that is tailor-built to implement these verification techniques for the discrete model. Through solving Boolean constraint satisfaction problems (SAT-based) and with guided stimulation (Genetic Algorithm), we can further extract the properties and behaviors of the system.

Furthermore, our model allows us to conduct cause-effect analysis of the apoptosis signal-

ing pathways. By constructing single- and double-fault models, we are able to study what fault(s) can cause the model to malfunction and the reasons behind it. Unlike simulation, our abstraction approach allows us to study the system properties and system manipulations from a different perspective without fully relying on simulation. Using these observations as hypotheses, we aim to conduct laboratory experiments and further refine our model.

To My Parents

Acknowledgments

Many thanks to all members of the Proactive group and friends who have helped and encouraged me throughout my graduate study. I want to express my sincerest gratitude and appreciation to my advisor Dr. Michael S. Hsiao for his support and his firm belief in me and my work. I would like to thank Dr. David Samuels for his active participation in this research in addition to guiding me and serving on my thesis committee. Furthermore, I want to thank Dr. Carla Finkielstein for spending numerous hours with me while her frogs are rampaging in her lab. I would like to extend my thanks to Dr. Lynn Abbott for serving in my thesis committee and once as my professor.

I especially want to thank Miss Jennifer Trinh for her help, encouragement and continuous support during my studies. Finally, I owe my deepest appreciation to my family especially my beloved parents for their love, selflessness and invaluable support during my life and studies.

HUY H. LAM

Virginia Polytechnic Institute and State University

June 2007

Contents

List of Tables

List of Figures

List of Acronyms

BDD ..... Binary Decision Diagram

BMC ..... Bounded Model Checking

CNF ..... Conjunctive Normal Form

CYTO ..... Cytoplasm

FSM ..... Finite State Machine

GA ..... Genetic Algorithm

GALS ..... Genetic Algorithm guided Logic Simulation

MAP ..... Mapping Automated Program

MITO ..... Mitochondria

nM ..... nano Molar

NS ..... Next State

ODE ..... Ordinary Differential Equation

PI ..... Primary Input

PO . . . . . Primary Output

PS . . . . . Present State

RA . . . . . Reachability Analysis

SAT . . . . . Satisfiability Problem, Satisfiable

UNSAT . . . . . Unsatisfiable

Chapter 1

Introduction

The growth of interdisciplinary research between engineering applications and biological science has led to many opportunities for the development of tailored tools and analytical frameworks for studying complex biological problems. Computational biology and bioinformatics have gained significant momentum between computer scientists and biologists in recent years. Computational biology, which relies heavily on computational modeling and simulations of software models, has been improving significantly due to the advance in computational power and inexpensive memory. Much of the research has been done in modeling the cell cycle, biochemical reaction pathway, and simulation of a subsystem in order to predict cell behavior as in [1]. The focus of these researches emphasize on building mathematical models and studying the simulation results that mimic the real world.

In our research, we used a different and unique approach to study a section of the apoptosis signaling pathways by constructing the pathways as a discrete hardware model. This introduced a unique level of abstraction by looking at the biological system as a *finite transition system*. Using this model, we applied testing and verification techniques to study the properties and behaviors of the model. One particular interest is to see if a certain protein concentration is achievable in the signaling pathway. If it is not achievable, we attempt to provide a proof that such protein

concentration is indefinitely not achievable given an initial starting condition. Such information can be extremely valuable to the understanding of apoptosis.

Computational biology can be inadequate in addressing the understanding of system properties and system behaviors. This is due to the fact that computational biology relies heavily on simulation and requires intensive computational resources. These approaches can be costly and impractical using simulation alone. However, verification techniques that we propose can be conducted with or without simulation. Spun off from hardware testing and verification research, we use currently available academic and industrial tools as well as internal developed tools to perform testing and verification on our specific apoptosis biological model. Our main focus is on property checking and reachability analysis to compute which proteins' concentration is achievable and which are not. The goal of reachability analysis is to determine if a given target protein concentration is realizable in the signaling pathway of interest. Furthermore, we performed fault model construction, analyzed the faulty behavior and interpreted the meaning of the faults. Such analysis can be extremely valuable to the understanding of apoptosis. Our observation can be used to form hypotheses for laboratory experiments. The future laboratory results will be used as feedback for the refinement of our models.

1.1 Programmed Cell Death: Apoptosis

Apoptosis is a controlled process in which a cell is pre-programmed to destroy itself if certain conditions are met. It is a highly orchestrated form of cell death in which cells neatly commit suicide by chopping themselves into bits. Apoptosis is critical to a multicellular organism's development and survival for removal of unnecessary, damaged, or aged cells without the inflammation often associated with necrosis, in which necrosis is referred as accidental cell death. This disorderly death makes it harder to clean up and often causes inflammation. As an orderly process, apoptosis is involved in sculpting the shape of the nervous system during development

and maintaining the normal functioning of the immune system. Defects in apoptosis therefore result in major developmental abnormalities. Apoptosis plays a role in the adult, for example, by contributing to proper turnover of cells in the skin or gut. The tight coupling of cell death and cell multiplication ensures in many tissues a constant, controlled flux of fresh cells, which are crucial to the preservation and optimal functioning of the adult organism. Malfunction in the coupling of apoptosis and cell multiplication result in pathologies such as tumors or functional deficiencies [2]. When there is too little apoptosis, it can lead to cancer and autoimmune disease. And when there is too much apoptosis that can lead to possibly for stroke damage or the neurodegeneration of Alzheimer's disease [3]. This delicate system of apoptosis has caught the attention of researchers worldwide for its critical role in the human body.

Apoptosis process involves a complex system of pathways of different protein signals. There are numerous independent pathways that can trigger apoptosis. Either through external signals transmitted to the cell's wall receptors or through internal signals activated by intracellular effectors proteins. We conducted our study on the internal apoptosis signaling pathway mediated by the mitochondria. Based on the current known model of mitochondrially mediated apoptosis signaling pathway, this complicated network of signaling reactions involve over 31 different proteins [4, 5, 6, 7, 8, 9]. This signaling pathway involves three major proteins: BID, BAD and BCL2. We used a well-known signaling pathway reaction database known as REACTOME [10] to build our finite transition system. Using this biological signal pathway, which will be described in detail later, we envision a modeling and analytical framework based on the proposed finite transition system model. With some modifications, the finite transition model can be easily made suitable for intracellular signaling pathway research. After the construction of the model, we used the analytical framework to performed numerous verification analysis and property checking techniques to learn about the model behaviors and properties. Such observations in the model can help us understand and infer behaviors for the real apoptosis system. Thus, we can assist the biologists by forming hypotheses based on these observations for laboratory experiments. The analysis includes

finding a sequence of stimulation that can bring the system to an apoptotic state or finding what fault or faults can make the system to become apoptotic when it is not supposed to and vice versa.

1.2 Design Verification

As mentioned above, our framework is built to apply design different verification techniques on our model. *Design Verification* is considered a process to verify that the model and implementation behaves the same as the specifications. This ensures that our model is consistent to the real apoptosis system. Design verification can be approached using two complementary techniques. First is the *Simulation-based verification* approach. This technique uses software as a simulation tool to exhaustively simulate (almost) all possible combinations to verify that the model result is as expected. However, simulating all possible input stimuli is impractical for large designs and limited by time resources. Second is the *Formal verification* approach. This technique uses mathematically reasoning to analyze the design. The technique can prove correctness of a circuit through Equivalence Checking or Model Checking and other approaches.

One approach in *Formal verification* is to convert the design into a Boolean formula. A Boolean formula is an equation of Boolean variables which can evaluate to either 1 or 0 signifying TRUE or FALSE, respectively. Similarly, a Boolean variable can only hold either a value 1 or 0. By converting the design into a Boolean formula, the technique can be easily modeled as a Boolean satisfiability problem (SAT) in hardware verification, where *Boolean Satisfiability* is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE. Equally important is to determine that no such assignments exist, implying that the function expressed by the formula is identically FALSE for all possible variable assignments. In the latter case, we would say that the formula is unsatisfiable; otherwise it is satisfiable. We used *Satisfiability* extensively to determine if a certain property can be satisfied given the model and some initial condition. Mostly, the main property that we focused

on is reachability, where we tried to determine if certain protein's concentrations is realizable or not by solving Boolean satisfiability problems. Note that we will be using realizable, achievable and reachable interchangeably in this thesis.

However, formal verification can face memory and time explosion problems for large circuits. As both simulation-based and formal techniques have their own merits, there are hybrids of formal and simulation methods that hope to combine the best of the two worlds.

1.3 Reachability Analysis

As mentioned above about reachability, there are numerous methods to perform *reachability analysis* besides using the SAT-base approach. *Reachability analysis* (RA) is to traverse the state space to determine whether each state is reachable or not. RA aims to reach as many states as possible by searching through the state space or by applying formal techniques. As an informal technique, simulation-based approach requires state space traversal to compute the set of reachable states from a given initial state. As for formal methods, many have been developed using binary decision diagrams (BDDs) symbolic state space traversal in [11, 12], SAT-based image computation in [13, 14], and the combination of both in [15]. Similar to SAT-based approach, where the design is converted to a Boolean formula, the BDDs approach uses a BDDs to represent the Boolean formula. This method compactly stores the Boolean logic of the design hence attempting to capture the entire design in a BDD. By traversing the BDDs, this verification approach indirectly traverses the design state space. However, building the BDDs is very time consuming and is prone to explosion in BDD size and memory.

In symbolic state space traversal using BDDs, image computation is performed to enclose all reached states from an initial state. Given a finite transition diagram in Figure 1.1, the first image set of S_o are all adjacent states that can be reached from state S_o by traveling across the arrow. The second image set of S_o are all states that can be reached from state S_o in two steps.

The image set of S_o after each iteration is shown in the Table 1.1. After the 4^{th} iteration, the image set stopped expanding. This is referred as a fixed point where the entire image set from S_o is captured. This set represents all states that can be reached when starting from S_o. Note that in symbolic traversal, a *set* of states is traversed instead of visiting one single state at a time to reduce the execution time.

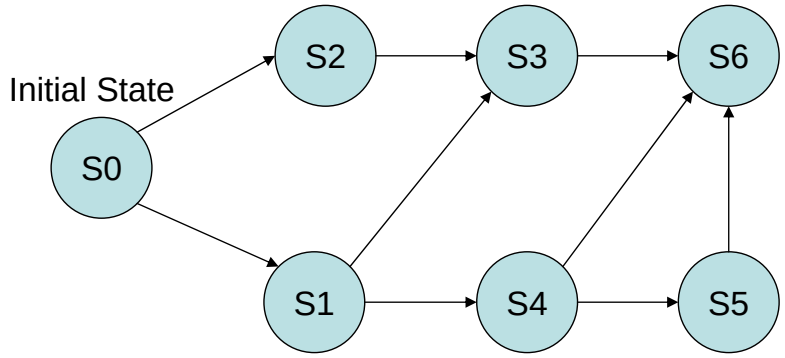


Figure 1.1: Finite Transition Diagram.

Table 1.1: Fixed Point Iteration in Image Computation

Iteration	State Reached
0	S0
1	S0,S1,S2
2	S0,S1,S2,S3,S4
3	S0,S1,S2,S3,S4,S5,S6
4	S0,S1,S2,S3,S4,S5,S6

When using the SAT-Based technique, it is prone to time explosion problems. SAT-based reachability analysis is modeled as a constraint satisfaction problem to compute the reachable states. A circuit is converted into a Boolean formula expressed in Conjunctive Normal Form (CNF). Each clause in the CNF is a sum of single literals, where each literal is a variable or its negation. An n-clause is a clause with n number of literals. The objective of SAT-based reach-

ability analysis is to find a set of assignments to the variables that can satisfy the CNF formula, or in other words, make the formula to evaluate to a logic 1. Formula 1.1 illustrates a small Boolean expression in CNF.

$$(\neg a + c)(b + \neg c + d) \tag{1.1}$$

$$(\neg a + c)(a) \tag{1.2}$$

This CNF has two clauses, with the first clause having 2 literals and second clause having 3 literals. The assignment that can satisfy this formula requires that *a = 0* and *b = 1*. Hence, we conclude that this Formula 1.1 is satisfiable (SAT). However, Formula 1.2 cannot be satisfied with any assignment, where *(a)* is called a uni-clause. Any assignment to variable *a* will cause at least one clause to evaluate to zero, hence making the whole formula zero. This formulate is unsatisfiable (UNSAT). When applying the SAT-based technique for reachability analysis, the circuit is converted to a CNF representation where the initial state and each target state defines additional constraints by adding uni-clause to the CNF. If an assignment can be found to satisfy this formula, then we can conclude that the target state is reachable from the initial state. Otherwise, if it is unsatisfiable, then this target state cannot be reached from the initial state. In our research, we will also discuss the use of SAT-based approach in bounded model checking and the use of SAT-based induction for our analysis.

1.4 Our Research & Contributions

This project is a joint research between Electrical and Computer Engineering, Virginia Bioinformatics Institute and the biology department at Virginia Tech. As under Electrical and Computer Engineering, our primary objective for this research project is to apply the current

knowledge from the discrete finite state system in the development of apoptosis system models richer than previous work. We aim to use this model to analyze the controllable and sensitivity behaviors with respect to the interaction with pathogens, drug delivery systems and experimental setup. Using mathematical predictions from the hybrid system and the discrete model, we can form hypotheses that can be carried out in laboratory based collaboration with the biology department.

The hybrid stochastic system model is done by a different student, Christopher Howells in our group, while the experimental lab work is carried through the biology department. The main focus on the research on discrete finite transition modeling is in developing the necessary model from an apoptosis signaling pathway. This discrete finite state system models three major proteins: BID, BAD, BCL2 individually as well as a unified model. The first part focused on constructing a healthy control model for each protein that reflects what is known of the signal pathway. Then, we combined the three subsystems to construct our complete, interacting model between the proteins. Next, we performed extensive analysis using verification techniques such as property checking and reachability analysis on the model. In the second phase, we applied different manipulations to the system and performed extensive stimulation to observe any deviation for the manipulated models. These manipulations include injecting single and double faults to the model. By studying the causes of these manipulations, we aimed to interpret the nature of the faults and form hypotheses for laboratory experiments.

1.5 Organization

The remainder of the paper is organized into five parts. Chapter 2 covers an in depth technical background of the apoptosis signaling pathway, in addition to various verification techniques used in this thesis. Chapter 3 focuses on the construction for both the subsystem and the complete system model. This chapter also presents the different verification analysis results using both formal and informal methods. Chapter 4 covers the different manipulated, faulty models and how

each affects the overall system. Chapter 5 sums up our work and discusses our future goals.

Chapter 2

Background

In this chapter, we cover in depth the technical background of the apoptosis signaling pathway. In addition, we will discuss the verification approaches, both informal and formal methods, which we developed and tailored to fit within this particular problem. This will include a detailed discussion of the construction of our Genetic Algorithm Guided Logic Simulation (GALS) tools as well as the SAT-based verification approaches. These two methods work orthogonally, where GALS is fast but incomplete, and SAT-based approach is complete but slow and faces time plus memory constraints. GALS is used to quickly identify all easy realizable protein concentration by using guided pseudo random stimulation that brings the system from some initial proteins' concentrations to a desire target concentrations, whereas the SAT-based approach is used to identify harder to realize target concentrations missed by GALS or provide proofs that such concentrations are un-realizable indefinitely.

2.1 Apoptosis Signaling Pathways

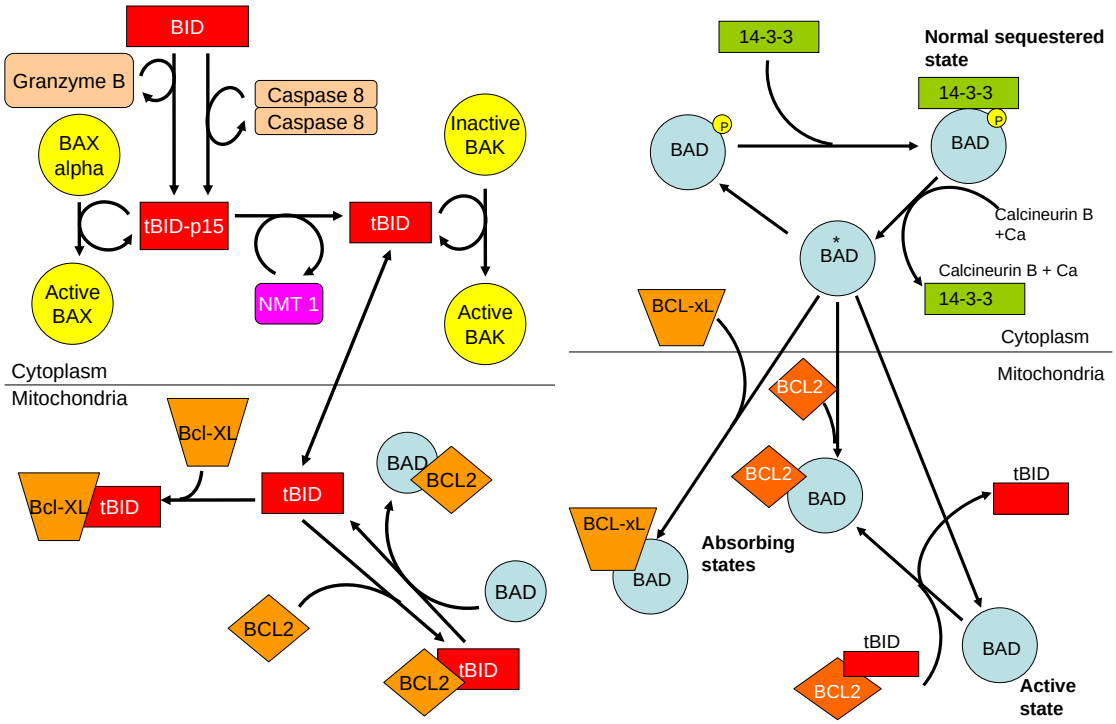
Apoptosis is known to be caused by numerous mechanisms in a cellular system through either external signals or intracellular proteins activation. The mitochondria have been known

to play a pivotal role in mediating the intracellular apoptosis pathways. There are at least three general apoptotic mechanisms known, and their effects may be interrelated [8]. Here, we studied one of the mechanisms mediated by the mitochondria. Individual elements of this mitochondria mediated signaling pathway can be found in all of the standard reaction databases. For this thesis, we have chosen to use the REACTOME database [10] as our basic initial source for the reactions of the apoptosis pathways. The REACTOME database currently lists 31 basic proteins involved in the apoptotic signaling in humans. Table 2.1 lists the elements of both the intrinsic and extrinsic signaling pathways and other non-protein modifiers. The REACTOME database information and research done in [16, 17, 18, 19, 20] will help us to construct and refine our initial model.

Table 2.1: Elements of the Apoptosis Signaling Pathways

Intrinsic Pathways (20 proteins)
BAD, BID, BAX, BAK, NOXA, PUMA, BIM, BCL2, BCL-xl, DLC1, DLC2, BMF, Cytochrome-c, SMAC, APAF-1, Caspase-9, Caspase-3, Caspase-7, Caspase-8, XIAP
Extrinsic Pathways (11 proteins)
FASL, FAS receptor, FADD, Caspase-10, TNF, TNF receptor, TRADD, TRAF2, RIP, TRAIL, TRAIL receptor
Controlling Factors
Granzyme-B, NMT1, Akt1, 14-3-3, Calcineurin B, P53, E2F1, MAPK8(JNK)
Significant Non-Protein Factors
Calcium, reactive oxygen species (ROS), ATP, cardiolipin, mitochondrial membrane potential

The complexity of this protein network is magnified because each of the 31 proteins can exist in a number of different state configurations. As such, a network is generated for each protein from the interrelating connections among these states, called a protein state network. where each protein state network interacts with one or more of the other protein state networks. Two examples are given in Figure 2.1, for the proteins BID and BAD [21, 22, 23]. Each of the two proteins have six separate configuration states: three in the cytoplasmic compartment and three in the mitochondrial compartment of the cell.



(a) BID System (b) BAD System

Figure 2.1: Two Sample Protein State Networks.

Each diagram illustrates the relevant configuration states. The asterisks denote the unmodified cytoplasmic proteins (from gene production). In the BAD diagram shown in Figure 2.1(b), we have superimposed some simple interpretations of some of the protein states. In the "normal sequestered state", BAD is awaiting activation (in this case by Calcineurin B and calcium). The "absorbing states" represent control mechanisms mediated by BCL-xL and BCL2 that deflect some BAD proteins from activation. Once activated and translocated to the mitochondrion, BAD can release tBID from its own sequestered form of tBID-BCL2 complex. The released tBID then translocates itself to the cytoplasm. Here tBID plays a critical role in the release of cytochrome-c and other factors from mitochondria [24]. Figure 2.1 illustrates another critical role of mitochondria in apoptosis. The mitochondria act as assembly sites for many of the pro- and anti-apoptotic factors [25].

In this section of the signaling pathway, we will be interested in studying the activation of BAD and how the concentration of tBID in the cytoplasm varies. Through performing reachability analysis, we can define what concentrations of tBID can be achieved and what stimulant is sufficient to achieve it. Orthogonally, SAT-based verifications are used to prove that certain tBID concentrations or any other proteins' concentrations are indefinitely un-achievable given a certain initial condition.

2.2 Finite Transition System

The protein state network diagrams above can be represented as a simple finite state machine (FSM). We chose this implementation approach to abstract a biological signaling pathways as a hardware circuit. Thus, we can utilize the strengths of hardware testing and verification techniques that allow us to study both the system properties and the system behaviors effectively without relying on full simulation. Each protein diagram is implemented as a FSM module representing the corresponding finite transition diagram in Figure 2.1. The three protein diagrams BID, BAD, and BCL2 are implemented separately and also combined as one interconnected FSM module to represent the whole apoptosis system.

Here, we will discuss the design of BID module. We note that the BAD and BCL2 modules are designed using the same concept. A FSM is defined by a six-tuple $\{I, O, S, s_0, \lambda, \delta\}$: a set of inputs, a set of outputs, a set of states, an initial state, an output function and a transition function, respectively. The inputs of the BID module contain all external substrates that assist the reaction. These include: Granzyme B, Caspase 8, NMT1, BcL_XL, BAD, and BCL2. The inputs are assumed to be fully controllable and mediate the transformation between protein types. The state of the FSM is defined as a combination of all proteins' concentrations in this network. As for the BID system, there are six protein state configurations. Together the concentrations of each protein define the state of this system. The concentration value of each protein type in BID

is stored in registers, which are simply sets of flip-flops. The number of bits used to represent the concentration determines its resolution and range. For example, if we use 8 bits, then there are a total of 256 possible values that can represent the concentration of a protein. If the unit of concentration is nano-molar (nM), then the resolution is 1 nM and the range is 0-255 nM. Hence, if there are 6 different proteins in the BID diagram; then there will be 48 flip-flops (FFs) in the BID module. The values of these 48 flip flops together define what state the FSM is currently in. The FSM transition from one state to another is defined by its transition relation function. The transition relation function describes how the value of each flip flop changes per unit of time. We can use a quick example of describing a traffic light signal as a FSM for illustration. For simplicity, we define two states for the traffic light, whether it is red or green. The input would be the pedestrian's push button, triggered by the pedestrian when he or she wants to cross the street. The output is the state of the FSM which is whether it is red or green. The transition relation function determines whether the light should be red or green in the next time unit. As for our apoptosis FSM, the biochemical reactions were used as the transition relation function to describe the molecular protein interaction and how the concentration value of each protein changes over time. All the interactions between proteins are described by the following class of mass-action Equation [26]

$$A + B \xrightarrow{k_f} C \tag{2.1}$$

$$\frac{dA}{dt} = \frac{dB}{dt} = -\frac{dC}{dt} = k_f * A * B \tag{2.2}$$

$$X \xrightarrow{k_f} Y \tag{2.3}$$

$$\frac{dX}{dt} = -\frac{dY}{dt} = k_f * X \tag{2.4}$$

where k_f is the forward rate constant. The reaction rate of all protein's interactions are described by those ordinary differential equations (ODEs). Equation 2.2 computes the amount of product substrate being created per unit of time and the amount of reactant substrate or substrates being consumed for the reaction per unit of time. The rate of a reaction is proportional to the concentration of both substrates A and B with respect to the rate constant k_f. Lastly, we defined the output of the FSM as any signals or register values that we would like to monitor such as the concentration of a protein.

Protein BAD and BCL2 modules are also modeled with the same approach. Notice that there are interactions between all three BID, BAD and BCL2 modules. Therefor, the output concentration of a protein in one module can be input to another module. For example, BCL2 concentration is an output from the BCL2 module, but its value is an input to the BID and BAD module. This interaction is illustrated in Figure 2.2

Our objective is to build individual modules and perform analysis on each model. Then, we continue to refine the model using the analysis results as a feedback method. Additionally, by performing analysis on each model, we aim to perform laboratory based experiments from our finding. In parallel, we plan to study the combined model where all three modules are combined and observe any different behaviors.

In this thesis, we used several tools both internally developed and other available academic tools. In the next section, we will discuss the Genetic Algorithm guided Logic Simulation (GALS) tool that was developed specifically for this research. In conjunction, we also used other formal verification tools such as zChaff SAT solver [27]. zChaff is a tool developed by Princeton University, which searches for assignments that can satisfy a Boolean formula expressed in CNF format. zChaff is currently the most well known efficient SAT solver. However, it is still prone to memory and time explosion problems for large circuit instances. On the other hand, GALS is a search tool that is designed to avoid both time and memory explosion problems but does not guarantee the result to be complete.

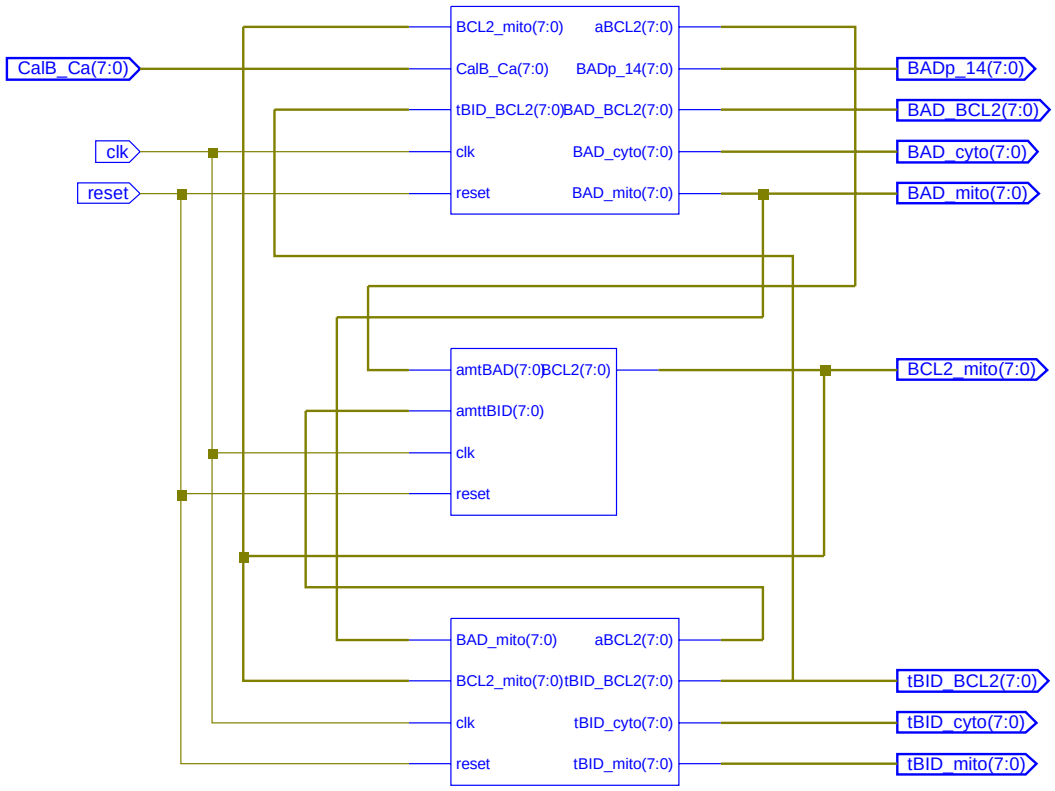


Figure 2.2: Verilog Combined Module Overview.

To be able to use all of these tools, the original behavioral Verilog module needed to go through numerous transformations to obtain the appropriate format for the verification tools. For example, GALS needed the design to be in a net list formal called .lev and zChaff SAT solver needed the .lev to be converted to a Boolean formula in CNF. Figure 2.3 below briefly describes the process. Documentation on how to setup and run the conversion process is included in the Appendix A.1. We used the simulation tool to quickly verify the correctness of the transformation process. Cadence logic simulation was used to capture the behavioral Verilog model's behavior. Then, we used our logic simulator in GALS to verify the same behavior for the .lev format. This assures that all intermittent formats were also properly transformed. In the next section, we will discuss the implementation of our GALS tool.

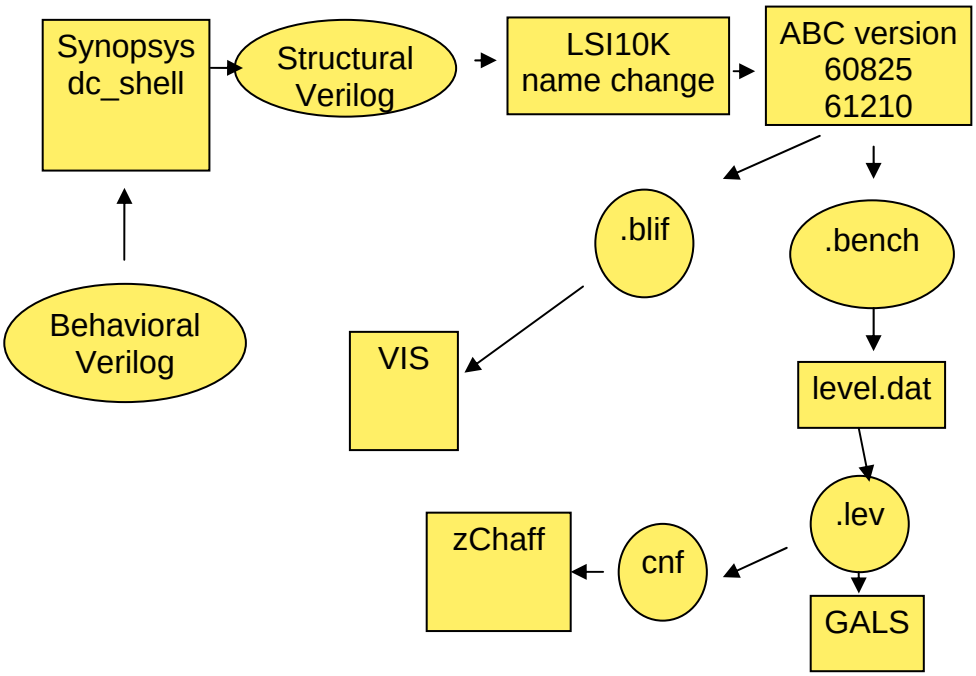


Figure 2.3: Conversion Flow Diagram.

2.3 Genetic Algorithm guided Logic Simulation

Reachability of a state in a sequential circuit representing the finite transition system has always been a hard problem to solve due to the potentially exponential explosion in time and memory. Consider a system with only 30 memory elements (e.g., flip-flops), there would be 2^{30}, or more than 1 billion, states. Formal methods have been proposed and implemented using Binary Decision Diagrams (BDDs) to compactly represent the states reached. BDDs have also been used to allow for symbolic traversal of the state space, allowing for enhanced model checking. However, BDDs are very sensitive to the underlying variable ordering and can cause explosion in memory. On the contrary, SAT based reachability analysis, even with significant improvements with SAT solver, faces time explosion. Both of these formal methods perform exhaustive search through the entire design space. Our approach used guided logic simulation to traverse through the search

space without causing either time or memory explosion problems. The primary objective of this approach is to target the reachability problem by answering the following question: Given an initial state, is there an input sequence that will bring the circuit to a given target state? In most circuits, including our apoptosis model, there are numerous easy to reach states. Using solely formal approaches can be overkill for those easy to reach targets. Hence, GALS is used to quickly explore the easy state space by simulation. Our approach used logic simulations guided by genetic algorithms and A* search with Hamming distance heuristic to find a sufficient sequence that brings the circuit to the specified target state.

2.3.1 Design implementation

The implementation of this tool is built upon an event-driven logic simulator and an A* search algorithm with Hamming distance heuristic that invokes the logic simulator. Later on, we added a genetic algorithm engine to guide the search for those harder-to-reach states. Figure 2.4 presents a general overview of the tool layout. A* search employs a logic simulator, a genetic algorithm engine and a random sequence generator to conduct the search.

The logic simulator is implemented as an event-driven logic simulator, where each sweep through the circuit represents a clock cycle in the circuit. Given a FSM and its starting state, the logic simulator simulates the circuit's behavior from that starting state using a set of primary input values for one time step. At each time step, the next state and the output of the circuit is evaluated. To assist the search process, the logic simulator provides methods to capture and load the current circuit gate values.

A general A* search is built upon a graph search in Algorithm 1. Given a starting state, it computes multiple next states by simulating the circuit at random for one time step; in hopes that the circuit will end up in a new state. Each next state's cost is computed using a chosen heuristic which is an approximate measure of how close this state is to the target state. The expanded states

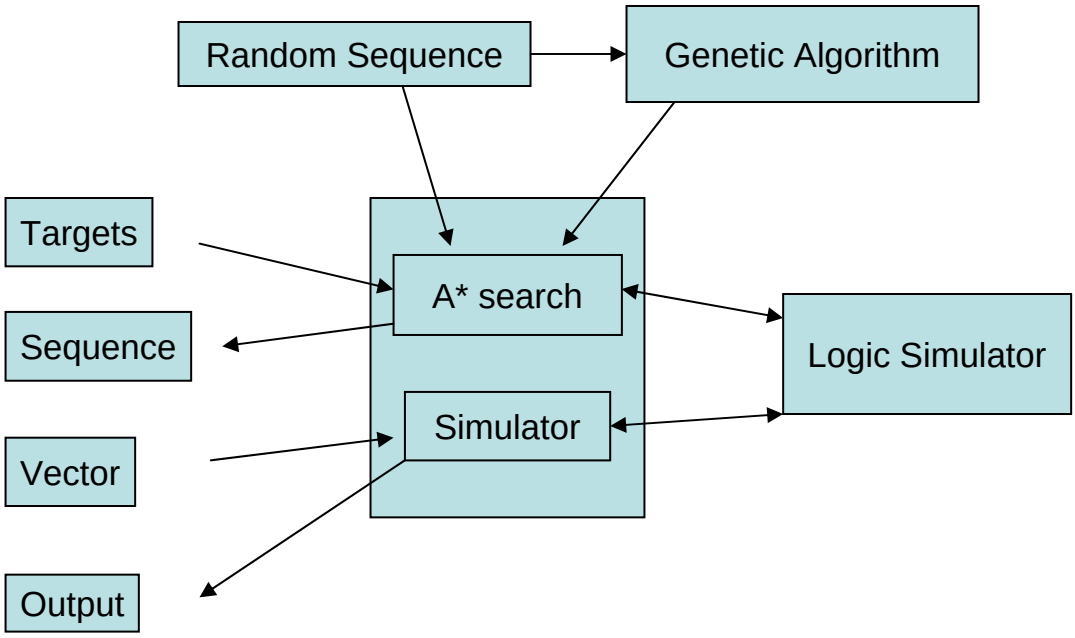


Figure 2.4: Genetic Algorithm Guided Logic Simulation Engine Overview.

are added to a priority queue sorted by their corresponding cost. The lowest cost state is popped out to be expanded next. The search continues until the target state is reached or an upper limit of node has been expanded. What makes a graph search an A* search is how the cost function $f(n)$ is computed. Let:

$g(n) =$ the cost of getting from the initial node to node n

$h(n) =$ is the estimate cost from n to target state using a given heuristic

A* defines: $f(n) = g(n) + h(n)$. This serves as the estimate cost for the best solution starting at initial state that goes through node n.

In our approach, the cost calculation is based on the state flip-flop values that we are targeting. The heuristic is based on Hamming distance, which counts the number of bits that are different in the current register's values to the target value.

Starting at the initial state, where all gates are set to "don't-care," the state is expanded by

Algorithm 1 Graph Search Algorithm

```
1: Function GRAPH-SEARCH(problem, fringe) return a solution or failure
2: closed ← an empty set
3: fringe ← Insert(make NODE(initial state(problem, fringe)))
4: loop
5:    if fringe is empty then
6:       return failure
7:    end if
8:    node ← remove-first(fringe)
9:    if GOAL(node) then
10:      return solution(node)
11:   end if
12:   if node is not in closed then
13:      Add node to closed
14:      fringe ← insert(EXPAND(node))
15:   end if
16: end loop
```

computing its images set. An image set is a set of all next states reachable in one time step from the current state. The image set is computed by using the logic simulator to simulate the circuit given a set of randomly generated primary input (PI) values and the current state value. Each circuit state is subjected to numerous logic simulations to try to capture as many unique next states as possible. Duplicate next state images are discarded. The set of next states are then compared to all previously expanded states, in the closed list, to insure uniqueness of these new states. The unique states are stored into a *node*. A *node* is a data structure used to hold the information about the current status of the circuit. The *node*'s cost is then computed using Hamming distance heuristic and it is inserted into the priority queue. The priority queue will sort each *node* by its cost. The least cost state will be chosen for the next expansion.

In Figure 2.5, states B and C are partial images of state A. After state A is expanded, both states B & C are inserted into the queue. Assuming state C has a lower cost than state B, C would be chosen to be expanded next. States E and D are images of state C. Assuming D has the lowest cost compare to E and B, D was chosen next. This random logic simulation quickly captures the

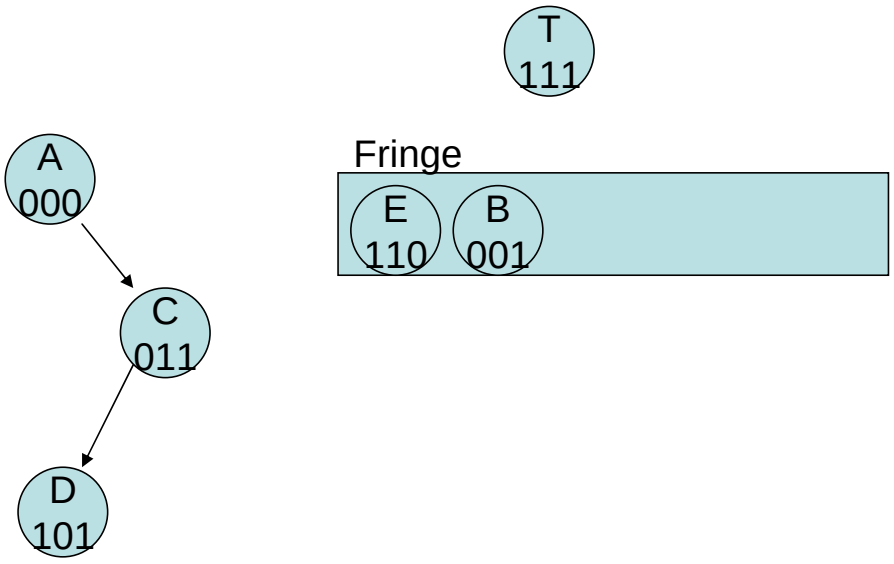


Figure 2.5: Reachability Search Using A* Search with Hamming Distant Heuristic.

image set without heavy computation as in formal image computation. But it is not guaranteed that all possible next states are captured.

Each *node* contains the circuit's current status information: current state, cost, parent state pointer and a child state point that defines the path to the target state. To be able to construct the sequences of the inputs that reached the target state, assuming that the search algorithm reached the target state, the node also stores the primary input values that is used to simulate the circuit from the parents' state to reach this state.

Node data members:

- Circuit gates' values
- Primary inputs' values that lead here from parent node
- Flip flop value of this state
- A parent pointer
- A child pointer

- Cost

Once the target state is reached, the algorithm traverses through the parent pointers from the target state to reach the initial state and forms a child linked chain from the initial nodes to the target node. By traversing this linked chain from the initial node, the sequence of primary inputs can be extracted to bring the circuit from the initial state to the target state. Figure 2.6 illustrates the concept of a search tree, where the black pointers are just regular parent pointers. The blue and red pointers form the child linked chain from initial to target node. All reachable states during the search from the initial state are included in the tree.

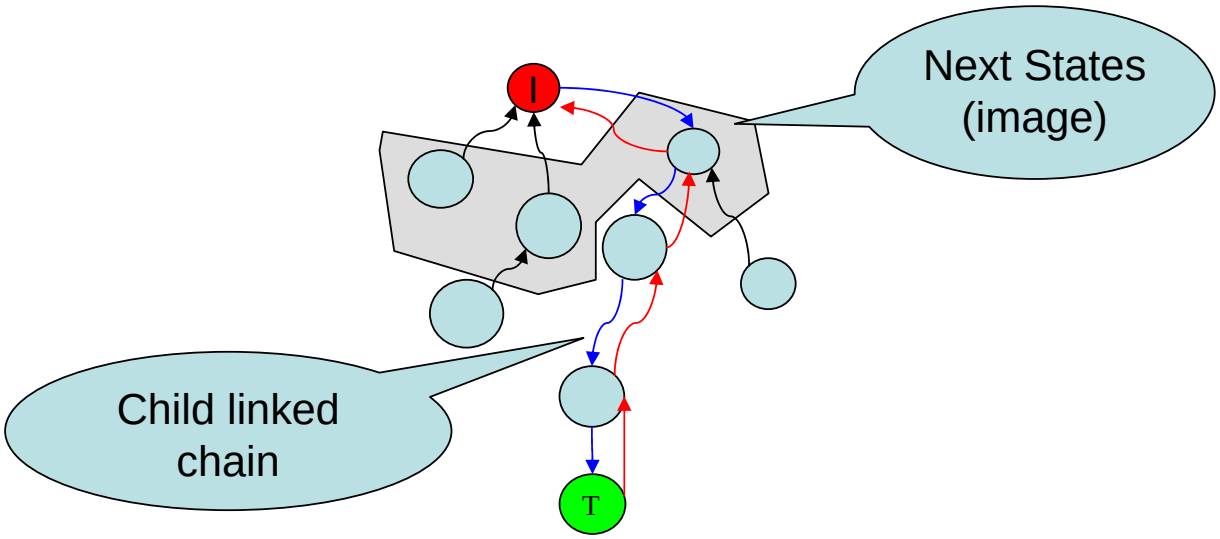


Figure 2.6: Reachability Search Tree.

2.3.2 Genetic Algorithm Search Engine

Random logic simulation searches can find many easy-to-reach states in little time. However, for the hard-to-reach states, doing a one time step image computation is not sufficient. The next image set that is unique may become empty, hence there are no more unique nodes to be

expanded. We implemented a Genetic algorithm (GA) engine to generate a more intelligent stimulus sequence that can reach hard-to-reach states and expand the search to a different region of the search space. The input sequence can simulate the circuit for more than just one time step. GA was first introduced by John Holland in his famous 1975 book [28]. GA is a search technique used to find true or approximate solutions for optimization and search problems. The technique is inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. GAs are implemented as a computer simulation to mimic the evolution process in nature. A population is composed of abstracted individuals containing chromosomes. Each individual carries chromosomes that are encoded as candidate solutions to a particular problem. The GA's evolution usually starts from a population of randomly generated individuals. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (crossover and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Our GA engine is implemented based on the simple GA described by Goldberg [29]. Many of the GA parameters, used for genetic operators such as population size, and mutation rate, are chosen based on [30] and based on some trial run on the model. Roulette wheel selection, two crossover and elitism were used for this framework.

Individual and population:

In our approach, the individual's chromosome is a sequence of primary input (PI) values that will be used to simulate the circuit. The initial population includes some random sequences as well as those good PI sequences seeded from the previous generation. The seeded individuals are chosen by looking at all reached states and select those states closest to the target state. Among those that are selected, the PI sequence is extracted to seed to the initial population. The basic genetic operators are covered below.

The crossover operator selects two parents using the roulette wheel selection and applies

the two-point crossover operation to produce two new individuals. Parent selection utilizes the roulette wheel algorithm, where the fitness of the individual directly correlates to its chance of being selected. The higher the fitness, the higher the percentage of being selected. This method gives the less-fit individuals a chance to be selected as opposed to the elitism method. Roulette wheel selection can be implemented as in Algorithm 2

Algorithm 2 Roulette wheel selection pseudo-code

1: Sum the fitness of all the population members. Call this TF (total fitness).
2: Generate a random number n, between 0 and TF.
3: Select the first population member whose fitness added to the preceding population members is greater than or equal to n.
4: Return the symmetric member on the other end with the middle of the population array as the center.

Two-point crossover is implemented by randomly selecting two pivot points in the chromosome. The genes between two pivot points are swapped to form two new genes. Theses two new genes are inserted into two new offspring individuals for the next generation. Figure 2.7 illustrates the operation of two crossover. Each new individual is subjected to a probability of a mutation set during initialization. If the individual is chosen, a random bit in the gene is inverted. Selection is the simplest operator in which the population is sorted from highest fitness to lowest. An N number of best individuals is selected from an M-sized population to survive until the next generation while $(M - N)$ individual perished to make room for N individual offspring. This method of selection is considered as elitism, which guarantees that the best individual will survive.

The fitness function plays a critical role to the evolution of the population. It acts as a guide to lead the population to converge to a targeted solution. A good fitness function leads the population well to its target, while a bad one can lead the population off target or to a local maximal point. Each new individual is evaluated to see the benefit of its genes in respect to the population. Hence, the fitness function is tailored to measure four qualities that can help reach the target state. These four qualities are weighted equally: circuit activities, Hamming distance, new

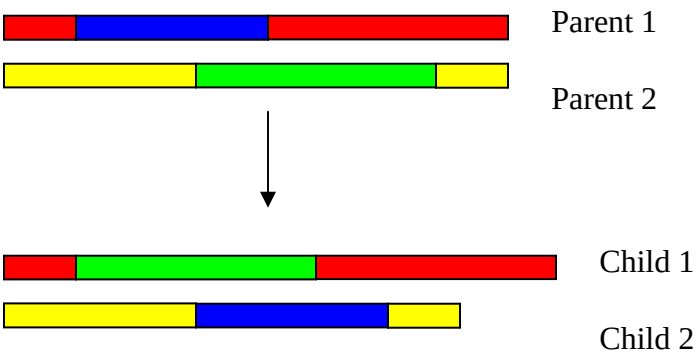


Figure 2.7: Two Crossover Illustration.

state reached, and target state reached. In [31], the approach was to measure both circuit activity and the number of flip-flops that changed its value for each input sequence applied. This method promotes high circuit activity and active flip-flop values but does not always lead to the target state. In our approach, we proposed the following fitness function. Let I be an individual and G be the gene that composes the sequence of input with S_i as a set of PI value applied at i^{th} time frame. Hence, gene or sequence $G = S_1, S_2, S_3, S_4...S_n$. The fitness value of a set of PI value S_i is f(S_i) = 0.25*circuit activity + 0.25*$\frac{\#FF-H(S_i)}{\#FF}$ + 0.25*(isNewState) + 0.25*(isTarget). H(S_i) is the Hamming distance between the circuit state after logic simulation using S_i and the target state. #FF is the number of flip flops in the circuit that is subjected to the search. The fitness of each S_n is subjected to the evaluation. The sum fitness values of all PI sets S_n is the fitness of the individual as defined in Equation 2.5.

$$F(I) = f(S_1) + f(S_2) + f(S_3) + ... + f(S_n). \tag{2.5}$$

Hamming distance quality is only awarded if the PI set leads to a new state and none if it reaches an

old state. This prevents developing a sequence that is repeatedly reaching old states to maximize fitness value. To promote a more compact sequence, a shorter sequence that leads to a new state using less set of PI should have high fitness than a longer sequence. We introduced a discount factor, γ. Given an individual I's gene sequence G of 4 of consecutive set S_i such that $G = S_1, S_2, S_3, S_4$. Now $F(I)$ is defined with γ in Equation 2.6:

$$F(I) = f(S_1) + \gamma * f(S_2) + \gamma^2 * f(S_3) + \gamma^3 * f(S_4) \; where \; \gamma = 0.7 \tag{2.6}$$

To further prevent the long, useless sequence that repeatedly reaches old states, an upper limit is set. So that after n number of consecutive old states reached, the rest of the sequence will not be evaluated and a low fitness value is assigned. This can save much computation time during fitness evaluation. However, when this limit is reached, the limit value is incremented by 1 for the next generation.

2.3.3 Hybrid Approach

Using random logic simulation alone cannot achieve high coverage for hard-to-reach states, and using GA alone without random logic simulation cannot achieve as many reachable states during the search process. The GA method is more direct and has a very high overhead in evolving the population, some states can be easily reached using random simulation, therefore using GA can be overkill sometimes. Hence, we combine both GA and random logic simulation in the search. Given a target, the engine's first attempt to expand N number of nodes using random logic simulation. If this does not reach the target state, then the engine invokes the GA to expand $N/2$ number of nodes.

2.4 SAT-based Verification Approaches

Using the preceding GALS, we can find a subset of reachable state space and a sufficient input sequence that can bring the system to the those reachable states. Because GALS is a simulation approach, it is incomplete and can miss some reachable states. Hence , we introduce SAT-based verification approach called SAT-based Bounded Model Checking that can target some of the state that GALS missed. During the search process, GALS has encountered numerous states that can not be reached. By using SAT-based induction, we can orthogonally prove some states as unreachable indefinitely.

2.4.1 SAT-based Bounded Model Checking

Model Checking is the process of checking whether a given model satisfied a given property or logical formula. In our application, SAT-based Model Checking is used to check whether a target state, or protein concentration, can be realized given the apoptosis model and an initial state. In this thesis, the interested properties are often associate with the realizability of a protein concentration encoded as a logical formula. Notice that we use properties, target states and protein concentrations interchangeably. When model checking is performed on a combinational circuit, since there are no flip flops, there is no concept of state or time. The combinational logic is converted directly to CNF. For a sequential circuit, the state elements add a time dimension to the circuit. Hence, we only discuss about bounded model checking, where a property is checked whether it can be satisfied for a given bound between points in time. At each clock cycle or time step, the snap shot of the sequential is treated as a time frame, similar to a movie frame. A bound of k time-frames is often referred as snap shots of length k of the sequential circuit. In our analysis, we are interested in checking if a given target state can be reached within a bound of length k time-frames. To convert a sequential circuit of k time-frames to a CNF, k different snap shots of the sequential circuit are concatenated back to back in a process called unrolling. The unrolling is done by placing copies

of the combinational elements of the circuit back to back for k times. The flip flops between time-frames are treated as buffers, each time-frames has its primary inputs and outputs. The signals that feed to the flip flops in the first time-frames are treated as pseudo primary input where initial state constraint is specified. Pseudo primary outputs hold the value of the last time-frame flip flops. This unrolling process is illustrated in Figure 2.8. After the sequential circuit is unrolled, it is treated as a combinational circuit and converted to a CNF for verification.

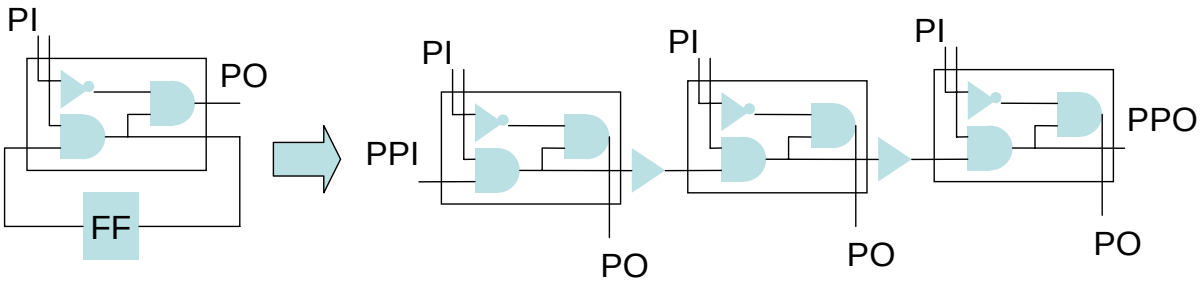


Figure 2.8: Sequential Circuit Unrolled for 3 Time-Frames.

Recent advances in Boolean Satisfiability has made SAT-based Bounded Model Checking (BMC) [13] more practical. In BMC, we unroll a sequential circuit for n time-frames, similar to an Iterative Logic Array expansion, shown in Figure 2.9. This unrolled circuit treats the present state in the first time-frame, PS(1), as pseudo primary input, which will hold the value of the initial state of the circuit. The unrolled circuit is then treated as a combinational circuit and converted to a formula in CNF form. The constraints of a target state is applied to the each NS(i) and the entire CNF is sent to a SAT solver in this case, zChaff [27].

Given a finite-transition system, let S_0 be the initial state(s), and $T(s, s')$ be its transition relation and ϕ be an invariant to be verified. An invariant is a property that hold in every time-frame in a given bound of the sequential circuit. For a reachability property, if the interested target state is encoded as a Boolean expression $\neg\phi$, and we can show that ϕ is an invariant, then we can conclude that $\neg\phi$ would never occur in this given bound. Hence, the target state is unreachable for

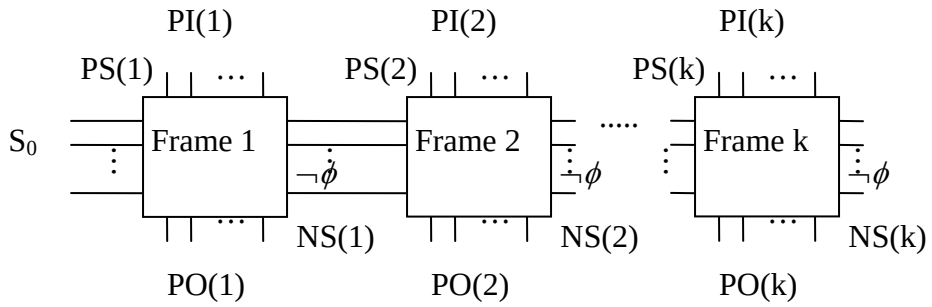


Figure 2.9: Iterative Logic Array Expansion for BMC.

the given bound of k time-frames. If ϕ is not an invariant, then $\neg\phi$ hold at some time-frame in the bound and the target state is reachable within that bound.

When a system is unrolled to an length k in Figure 2.9, T_i and ϕ_i denote the transition relation and the property assertion at step i, respectively. To verify if the interested ϕ is an invariant, we inject $\neg\phi$ at each time-frame step i. Then, the formula for the unrolled instance with the property monitor can be constructed by $\neg\phi^*(1,2,...k) \wedge T^*(1,2,...k)$ where $\neg\phi^*(1,2,...k) = \neg\phi_1 \vee \neg\phi_2 \vee ... \vee \neg\phi_k$ and $T^*(1,2,...k) = T_1 \wedge T_2 \wedge ... \wedge T_k$. $\neg\phi$ is a constraint applied at all time-frames. zChaff is used to try to satisfy this Boolean formula. In order to speed up the process, we utilized available CNF preprocessing tools such as Hypre [32] and NiVER [33]. However a bug was found for the NiVER tool, hence only Hypre is used for the preprocess computation. Hypre tools reduced the size of CNF up to 50% and reduced run time of zChaff up to 80%. If the CNF formula is satisfiable (SAT), we can conclude that $\neg\phi$ holds at some time-frame from the initial state I, which means ϕ is not an invariant. If the formula is unsatisfiable (UNSAT), we know ϕ is a k-invariant (unreachable within k time-frames from the initial state I), but there is no conclusion made about the global invariance of ϕ. Hence, we would need to unroll the circuit deeper.

We also implemented incremental BMC. When an unrolled circuit of length k is UNSAT, we need to unroll $n + k$ time-frames where $n > 0$. If we use the above approach, for each

time-frame from 1 to $n + k$, $\neg\phi$ will be injected as a constraint. However, since we know the circuit unroll up to length k is UNSAT, which means from time-frame 1 to k, $\neg\phi$ does not hold in any time-frame. Hence, from time-frame 1 to k we inject ϕ and from $k + 1$ to $k + n$, we still inject the constraint $\neg\phi$. The Boolean formula for this unrolled instance can be constructed by $\phi^*(1, 2, ...k) \wedge \neg\phi^*(k + 1, ...n) \wedge T^*(1, 2, ...n)$ where $\phi^*(1, 2, ...k) = \phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k$, $\neg\phi^*(k + 1, ...n) = \neg\phi_{k+1} \vee ... \vee \neg\phi_n$ and $T^*(1, 2, ...n) = T_1 \wedge T_2 \wedge ... \wedge T_n$.

2.4.2 SAT-based Induction

We used SAT-based induction to prove if a certain property is indefinitely unsatisfiable. In the context of our model, we can prove a certain protein's concentration is indefinitely unachievable starting from a given initial state. SAT-based induction is based on mathematical induction. In the base case, the circuit is unrolled for k time-frames and a conventional BMC is used to check for its unsatisfiability. In the induction step, the circuit is unrolled for k+1 time-frames, with the first k time frames constrained to hold the property. After converting each unrolled circuit to CNF expressions respectively, if we can show that for k+1 time-frames the property is unsatisfiable, then we can conclude that the property is unsatisfiable indefinitely in this model.

Let S_0 be the initial state(s), $T(s, s')$ be its transition relation and ϕ be an invariant to be verified. In SAT-based induction [34, 35], the base case $S_0 \rightarrow \phi$, also can be written as a Boolean formula of $S_0 \wedge \neg\phi$. If the base case is SAT, then ϕ is clearly not an invariant. If the result is UNSAT, then we need to perform the induction step $\phi_1 \wedge T_1 \rightarrow \phi_2$, which also can be written as $\phi_1 \wedge T_1 \wedge \neg\phi_2$. In the induction step, the initial state S_0 is left unconstrained. If the formula is UNSAT, then ϕ is proved to be an invariant for all possible initial states. However, simple induction is insufficient to prove most of the hard invariants. Thus, we introduce the use of strong induction also called induction with depth, which was used in [34]. Here, the base case is $S_0 \wedge T^*(1, 2, ...k) \rightarrow \phi^*(1, 2, ...k)$, where S_0 is the initial state, $\phi^*(1, 2, ...k) = \phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k$,

and $T^*(1,2,...k) = T_1 \wedge T_2 \wedge ... \wedge T_k$. The induction step is $\phi^*(1,2,...k) \wedge T^*(1,2,...k+1) \rightarrow \phi_{k+1}$. Both of these are converted to Boolean formula and are checked using zChaff for satisfiability. As for the base case, it is performed exactly as described in the SAT-based BMC section with S_0 as its initial state and $\neg\phi$ is injected in every time-frame. If the base case is SAT, we can conclude that ϕ is not an invariant. Otherwise, we need to perform the induction step. The unrolling of the circuit based on the induction step is shown in Figure 2.10, where Boolean formula $\phi^*(1,2,...k) \wedge T^*(1,2,...k+1) \wedge \neg\phi_{k+1}$ can be extracted from the unrolling. From time-frame 1 to k, ϕ is applied and $\neg\phi$ is applied at time-frame k+1. If this induction Boolean formula is UNSAT, then we can conclude that ϕ is an invariant indefinitely. Otherwise, if the Boolean formula is SAT, then nothing can be concluded, therefore we need to unroll the circuit to a higher depth k' and the process is repeated.

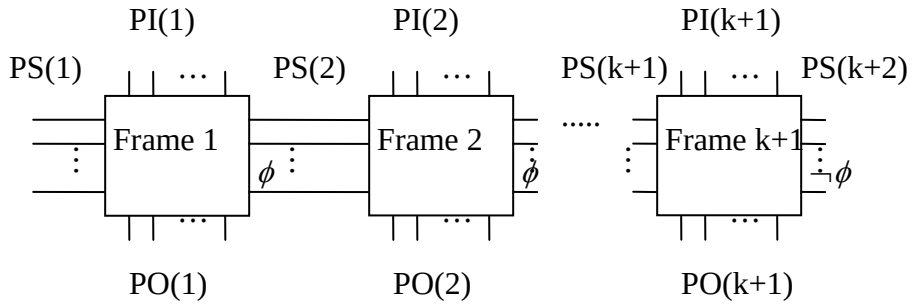


Figure 2.10: Iterative Logic Array Expansion for Induction.

An application for both SAT-based BMC and SAT-based induction can be used for a given BID model. The property we aimed to check is reachability. Let ρ be the property that defines the set flip flop value that specifies the target state we are interested in. Let $\phi = \neg\rho$. By SAT-based induction base case, we can prove whether ϕ is not an invariant for k time-frames. Similar to BMC of k time-frames, if the Boolean formula $S_0 \wedge T^*(1,2,...k) \wedge \neg\phi^*(1,2,...k)$, where S_0 is the initial

state, $\phi^*(1,2,...k) = \phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k$, and $T^*(1,2,...k) = T_1 \wedge T_2 \wedge ... \wedge T_k$ is SAT, then ϕ is not an invariant. This means $\neg\phi$ or ρ does hold at some time-frame n, where $0 \leq n \leq k$. If the formula is UNSAT, no conclusion can be made. We then use induction and unroll the circuit for $k+1$ time-frames. In the induction case, if the Boolean formula $\phi^*(1,2,...k) \wedge T^*(1,2,...k+1) \wedge \neg\phi_{k+1}$ for the is UNSAT, then ϕ is an invariant for the entire design, which means that ρ does not hold indefinitely. Using these two approaches, we can prove reachable state and unreachable state space of the design by intelligently specified ρ.

Chapter 3

Finite Transition System and Analysis

Our preliminary model was aimed to capture a rough behavior of the BID diagram. We studied the behavior and properties extensively on the BID model during the phase I of our study. Concurrently, a continuous model was being developed with laboratory based experiments literature to supply any necessary parameters and constraints we used for our model. As described above, all biochemical reactions are described by the differential Equation 2.2, where the rate constant k_f is unique for each reaction. We searched extensively in the literature of previous research on the rate constants. Many are unavailable and even contradictory. However, [26] provided us some useful rate constants for the BID model.

3.1 Technical Design of BID Model

The design of BID went through numerous iterative refinements from the initial model described in Chapter 2. Based on the range of each proteins in the BID module, we decided that a tenth of a nano molar (nM) will be the base unit of concentration for all substrates proteins. We studied our BID model using an 8-bit resolution for the concentration. Hence, the range for each protein is between 0.0 and 25.5 nM. From Equation 2.2, we must have the rate constant value, k_f,

for each reaction pathway of the mode. After multiple attempts to find the rate constants based on [26, 36], we have decided on using the flowing rate constants in Table 3.1.

Table 3.1: Reduced BID Model Rate Constants.

	Rate Constants k_f (unit)		
Reaction	Rate constant value	In discrete system unit step	Adjusted value for shifted multiplier
$\frac{d[mito]}{dt} = k_f[cyto]$	10.0 s^{-1}	0.3333 ($\frac{1}{30}s^{-1}$)	85
$\frac{d[cyto]}{dt} = k_f[mito]$	1.00 s^{-1}	0.0333 ($\frac{1}{30}s^{-1}$)	9
$\frac{d[tBcl2]}{dt} = k_f[mito][Bcl2]$	1.00 $nM^{-1}s^{-1}$	3.33x10^{-3}($\frac{1}{10}nM^{-1}\frac{1}{30}s^{-1}$)	218
$\frac{d[mito]}{dt} = k_f[tBcl2][BAD]$	1.00 $nM^{-1}s^{-1}$	3.33x10^{-3}($\frac{1}{10}nM^{-1}\frac{1}{30}s^{-1}$)	218

Due to the fact that there are inconsistent rate values for the two sources [26, 36], we chose to use straightforward rate values above for our first study of the BID model. The selected rate constant assumes all reactions to have the same magnitude but the translocation of tBID from the cytoplasm to the mitochondria is 10 times faster as seen in the second column. The values in the third column represents the rate constant in the unit step of $\frac{1}{30}$ second and $\frac{1}{10}$ nM. The last column are the adjusted values that are used to represent the rate constant in the discrete transition system. These adjusted values are computed based on the resolution of the implementation and the type of reaction. The adjusted values in the first two rows are derived by multiplying its respective third column's values by $2^{resolution}$ or 2^8. The adjusted values for the last two rows can be obtained similarly by multiplying its respective third column's value by 2^{2*8}. This is done because a custom multiplier was used in calculating the reaction rate as explains below.

As mentioned above in Chapter 2, all reactions were modeled using ODEs. To compute the reaction rate for each ODE, we had to use numerous multipliers. Minimizing the complexity and cost for the multipliers can reduce the logical net list implementation significantly. Initially, the concentration of each protein was designed to represent a percentage of protein concentration. For a 4-bit resolution protein, a value of 8 means $\frac{8}{16}$ signifies a 50% concentration. Since concentra-

tion used are fractional numbers, multiplication of two fractional numbers will always result in a smaller fractional value. Instead of using the traditional 4×4 bit multiplier, which returns an 8-bit result, the 4 least significant bits are truncated. The 4 most significant bits would be the fractional value numerator of the multiplication. For example: $\frac{4}{16} * \frac{8}{16} = \frac{32}{16^2} = \frac{2}{16}$, where the value 4 and 8 are stored as proteins' percent concentration. When multiplying these two fractions, we would expect $\frac{2}{16}$ to be the result, where the value 2 represents a percentage concentration. In fact, when 32 is shifted to the right 4 times, it is exactly 2. In other words, if 32 is represented by an 8 bits value and the last 4 least significant bits are truncated, it becomes 2. Due to the truncation of the half lower significant bits, some rounding error can accumulate. Hence, each multiplier is equipped with an adder that rounds up the result if it is equal or greater than 0.5 and rounds down otherwise.

In the latest iteration of BID model, the concentration was no longer representing a percentage but was used as an actual concentration in a tenth of a nM. However, the modified multiplier was still very useful to accommodate the very small magnitude of the RATE constants. For example, given a rate constant of 1 $nM^{-1}s^{-1}$ or $3.33 \times 10^{-3} \frac{1}{10} nM^{-1} \frac{1}{30} s^{-1}$ and resolution of r, each multiplication will shift the result to the right by r, which is equivalent to dividing by 2^r. To compensate for the shift, we multiple the rate constant by 2^r. For first type of differential Equation 2.4, the compensation only needs the rate to multiply by 2^r. For the second type of differential Equation 2.2, there are two multiplications. Hence, the rate is multiplied by 2^{2*r} resulting in $218 \frac{1}{10} nM^{-1} \frac{1}{30} s^{-1}$. The adjusted rate values for all four reactions in BID are shown in the right most column of Table 3.1.

Each protein concentration is represented by a register, which is simply a set of flip-flops. At each time step, the value of this set of flip flops are reevaluated based on how much was used up for reaction or how much was added to the concentration. Our model needs to consider underflow and overflow of the concentration value when performing additions and subtractions. By using the carry out bit of an adder and a subtracter, we can detect when an overflow and an underflow occurs, respectively. When an overflow or an underflow is detected, we cap the value at its max or

min, respectively. Notice that all the arithmetic in the model is treated with unsigned arithmetic, and all the registers and wires are unsigned. For an addition operation, if the adder's carry out bit is set, then the resulting sum is capped at its max value, where all bits are one. Similarly, for a differentiation, if the subtracter's carry out bit is set, then the resulting different is capped at zero.

Our model also considers when a protein belongs to two independent reactions, in which both reactions compete for the same protein. For a reasonable solution that does not add a huge complexity to the design, we kept a cap to how much each reaction can consume per time step. If both reactions have the same strength, each reaction can only take at most half of the protein concentration per time step. For one case in the BID model, the translocation from tBID_mito tBID_cyto is capped to take at most $\frac{1}{4}$ of tBID_mito and tBID BCL2 reaction takes the rest $\frac{3}{4}$. This is because the tBID, BCL2 reaction is three times stronger than the translocation of tBID.

3.1.1 The Reduced BID model

In a biological model, any reaction can be classified as a slow or a fast dynamic reaction. In our model, creation and decay reactions are considered to have slower dynamics, while protein reactions and translocation are considered to have faster dynamics. This is a standard stiffness problem in trying to accommodate both types of reaction. However, when we considered taking into account of the decay and creation reactions of a protein over time, the decay and creation amounts were insignificant in comparison. This is because of the small time step and the short time scope of the analysis where it is ranging between $\frac{1}{10}$ of a second to a few seconds. We chose to omit both of these factors. Because of the omission, the analysis needs to be initialized to a steady state protein concentration value. We used both Copasi simulation tool and ODEs from Christopher Howell's studies and found the same steady state concentration for each protein given that the creation rate of BID is 0.003 $\frac{nM}{s}$ and decay rate of 4x$10^{-5}\frac{1}{s}$. However, for a steady state, of which most of the BID protein complex is in the form of tBID-BCL2, our reduced BID model

had to have the tBID Bcl-XL complex removed. The resulting steady state concentration is listed in Table 3.2

Table 3.2: Reduced BID Model Initial Condition.

Non-apoptotic condition	nM
BID	0.001
tBID-p14	0.001
tBID_cyto	0.025
tBID_mito	0.250
tBID_BCL2	25.00

Due to this simplification, at the initial non-apoptotic steady state, there is no protein concentration present in the form of BID and tBID_p14 complex. Hence, our model did not include those two proteins. Therefore the Caspase8 and Granzyme B triggering pathways were omitted. The only way for apoptosis to occur is by the release of BAD protein that breaks up the tBID_BCL2 complex and releases tBID in the mitochondria. tBID in the mitochondria then translocates itself to the cytoplasm. Using this reduced model, illustrated in Figure 3.1, we applied GALS and simulation to find a sequence that targets a high concentration of tBID_cyto. In the BID model, BCL2 and BAD are two fully controllable inputs. The sequence that drives the system to a high concentration of tBID_cyto has a low concentration of BCL2 and a high concentration of BAD, which is intuitive. Drugs that lower the level of BCL2 have been known as anticancer agents [36], which trigger apoptosis. This result can be confirmed using the analytical approach to help verify the correctness of our model. Using our analytical approach, where BCL2 concentration was kept at zero and BAD was unlimitedly available, the highest concentration of tBID would be 2.3 nM when starting from the above initial state from Table 3.2. Our GALS tool was able to find sequences that reach between 0.0 to 2.3 nM tBID concentration.

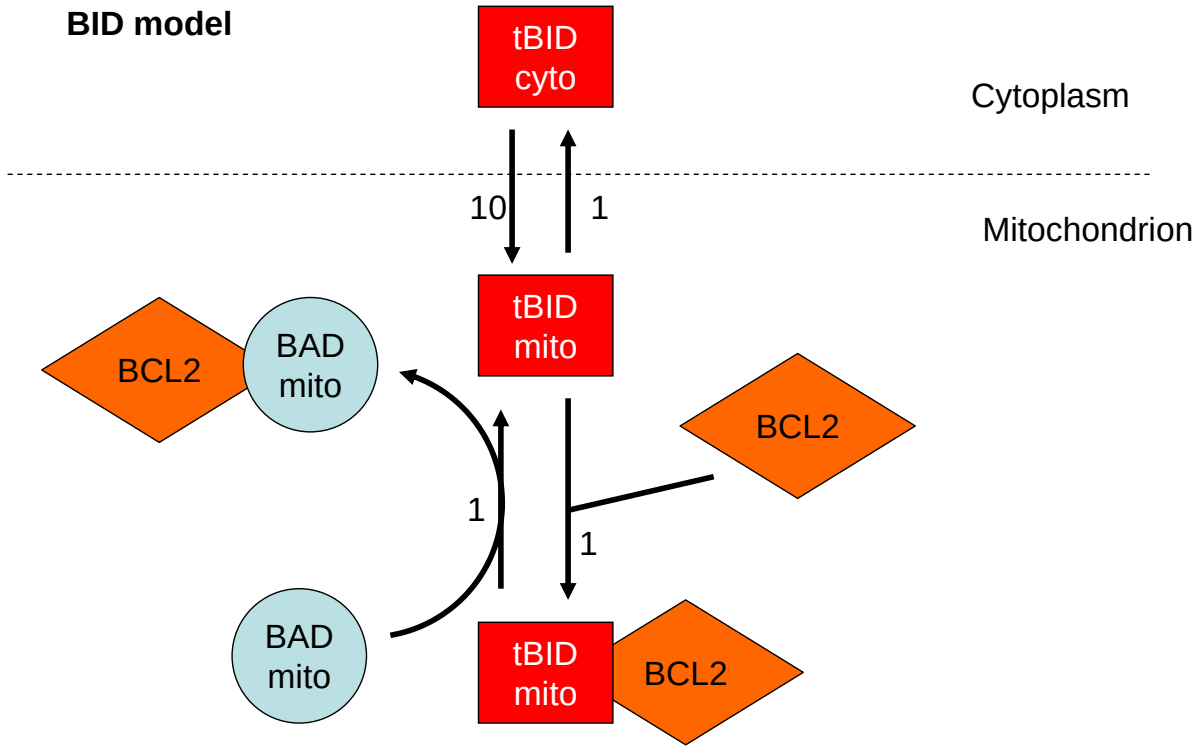


Figure 3.1: Reduced BID Protein Signaling Pathway.

3.1.2 Analysis of BID model overview

After constructing the reduced BID model, we used it as a preliminary study to apply different analysis techniques we discussed earlier. GALS was used in conjunction with formal SAT-based methods to determine both reachable and unreachable states. The GALS tool is fast but incomplete. It avoids both memory and time explosion problems. GALS can find sequences quickly to reach a target state by searching in a limited state space, while SAT-based method can take a long time because it considers the entire state space. We mainly used GALS to find as many reachable states and SAT-based induction to prove unreachable states, illustrated in Figure 3.2.

All analysis mentioned here and below were performed on a cluster of 15 computers, where each computer in the cluster has a Pentium D CPU running at 3.0 GHz with 2 GB of RAM. Using

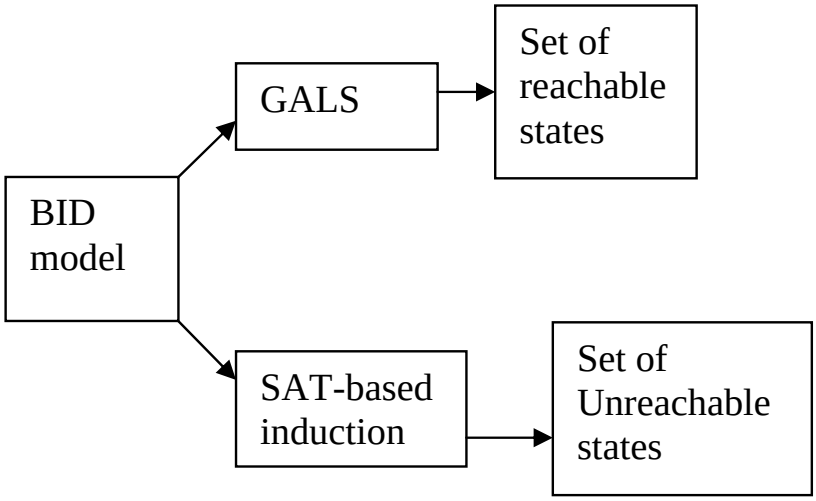


Figure 3.2: Model Verification Approach Overview.

the parameters specified in Table 3.3, GALS quickly identified many reachable states for the three proteins in the BID model in Table 3.4. For each protein, there are 256 possible state. The table

Table 3.3: Cluster Specifications and Experimental Parameters.

Specifications			
CPUs	Pentium D	Speed	3.0 GHz
Memory	2.0 GB RAM	OS	Redhat Enterprise Linux 4
Parameters			
Population Size	10	Number of Generation	8
Number of Seeds	5	Number to Breed	10
Node Expand Limit	600	Mutation Rate (%)	6
Max Image Size	20	Original Target	Yes
Search Rate	No	Hybrid	Yes

listed the number of reachable states out of 256 for both random input simulation and genetic algorithm guided simulation. By our analytical methods, we showed that tBID_cyto can only reach a highest concentration of 2.3 nM when BAD is at its max value of 25.5 nM. This helped us to verify that GALS had found all reachable states for protein tBID_cyto. The two different

input sequences for tBID_cyto that reached two different final concentrations are available in the Appendix B.1.1. Below, SAT-based induction method was used to further verify unreachable state space for each protein.

Table 3.4: Reduce BID Model Reachability Results.

		Out of 256	
Protein	Initial	Random	Genetic
tBID_cyto	0.0 nM	1	**24**
tBID_mito	0.3 nM	239	**239**
tBID_BCL2	25.0 nM	248	**250**

3.1.3 SAT-based BMC and SAT-based Induction

Given the reduced BID model, for those states that cannot be reached by GALS tool, we first attempted to use SAT-based BMC to find any more reachable state that GALS might have missed. If SAT-based BMC cannot show those states as reachable, SAT-based induction was used to provide a proof that those states are indefinitely unreachable. We used a stronger case of induction, called induction with depth, where in the base case, the BID model is unrolled for 14 time-frames. This means that we are considering about $\frac{1}{2}$ second of the dynamic. For the base case, the larger the time scope of consideration, the more likely that the induction case would be able to prove that a certain state is unreachable. However, the Boolean formula size increases linearly with the number of time-frames but the complexity increases exponentially. Hence, for any time-frame larger than 15, the Boolean formula becomes too complex in the induction case for zChaff to solve in a reasonable amount of time. With 14 time-frames for the base case, the target state and initial states are injected as properties and we performed SAT-based BMC. This step found 3 additional tBID_BCL2 concentrations as reachable. For those concentrations that are unreachable within this 14 time-frames bound, we unrolled the circuit for 15 time-frames and injected the properties as illustrated in Figure 2.10 with initial state left open. The SAT-based induction was able to

show proofs of many unreachable cases. However, for tBID_BCL2, it was not able to prove any concentration between 25.2 nM to 25.5 nM as unreachable. For this BID model, because there is no creation of tBID_BCL2, this protein can almost only increase to 25.3 nM if all tBID_mito combine with BCL2 to form tBID_BCL2 complex. This can be shown using a simulation tool by setting BAD concentration as zero and providing unlimited BCL2. Based on that knowledge, we concluded that two more states, 25.5 nM and 25.4 nM, for tBID_BCL2 are unreachable. Table 3.5 summarizes the analysis results for the BID model. For those states that are specified as unknown, the zChaff was not able to solve after 600 seconds time limit.

Table 3.5: Reduce BID Model Verification Results.

Protein	I (nM)	# R	# UR	# UNK
tBID_cyto	0.0	24	229	3
tBID_mito	0.3	239	5	12
tBID_BCL2	25.0	253	2	1

3.1.4 State Space Partition

Both SAT-based BMC and SAT-based induction were quite effective in providing conclusive results for the BID model when a target state is specified in sets of protein concentration. We observed that there is a very high correlation between the flip-flops. First, it is between the flip-flops that belong to the same protein concentration; second, it is the relationship between proteins in their respective reactions. Instead of partitioning the flip-flops by protein concentration, finding a partition group of flip-flops that can take advantage of the correlation can yield some interesting properties of the system. In an earlier work, [37] showed promising results in logic-simulation based test generation using static partition. While [38] illustrated the power of dynamic partition of flip-flops for sequential ATPG. More recently, [39] took a step further in partitioning internal gate signals for effective search space traversal. Given that we have good domain knowledge about the nature of the circuit, we can manually select a few partition targets that systematic methods

used in [37, 38, 39] might not have found.

Similar to the SAT-based induction approach in the previous section, the base case considered a little more than $\frac{1}{2}$ second dynamics or 19 time-frames of the circuit. Anymore time-frames would cause a time out for SAT-solver zChaff after 1200 seconds and any less time-frames would result in a less number of successful proofs in the induction case. By using the domain knowledge of the system, we targeted a few sets of partitions that helped us to learn about the sensitivity and the properties of the system. For example, we studied if the concentration of tBID_cyto and tBID_BCL2 can be high simultaneously, and if so, how high? If the two most significant bits of tBID_cyto and tBID_BLC2 are grouped as a partition, SAT-based induction can prove that some combinations are unreachable. Hence, we can know about the relationship between tBID_cyto and tBID_BCL2. Due to the simplicity of the stand alone BID model, we only present the relationship between tBID_BCL2 and tBID_cyto. Other relationships did not yield any interesting behaviors.

For tBID_cyto, we know it can only reach the highest concentration of 2.3 nM, hence any concentration between 1.6 nM and 2.3 nM is considered as high. Thus, we'd like to know if tBID_cyto is constrained at that range, what concentration of tBID_BCL2 cannot coexist. We increased the concentration of tBID_BCL2 from 24.7nM to 25.5 nM. After unrolling the circuit for 19 time-frames in the base case and 20 time-frames in the induction case with time out for zChaff at 1200 seconds, the result was not very promising. SAT-based induction was only able to prove that if tBID_BCL2 concentration is 25.5 nM then tBID_cyto cannot reach above 1.6 nM. We attempted the same approach between tBID_mito and tBID_cyto concentration. SAT-based induction proved that if tBID_mito concentration was above 25 nM then tBID_cyto cannot reach above 1.6 nM. This partition approach is not as effective for the BID model since many elements are not constrained. More effective result can be obtained in the full apoptosis model below.

3.2 Complete Apoptosis Model and Analysis

Based on the reduced BID model, we incorporated in a reduced version of both BAD and BCL2 models for a more complete model. In BAD, Bcl-XL was also removed in addition to the removal of BADp protein. BCL2 exists only in the mitochondria form and BIM, NOXA, PUMA were removed. Figure 3.3 is the full diagram of all three proteins. The design of BAD model

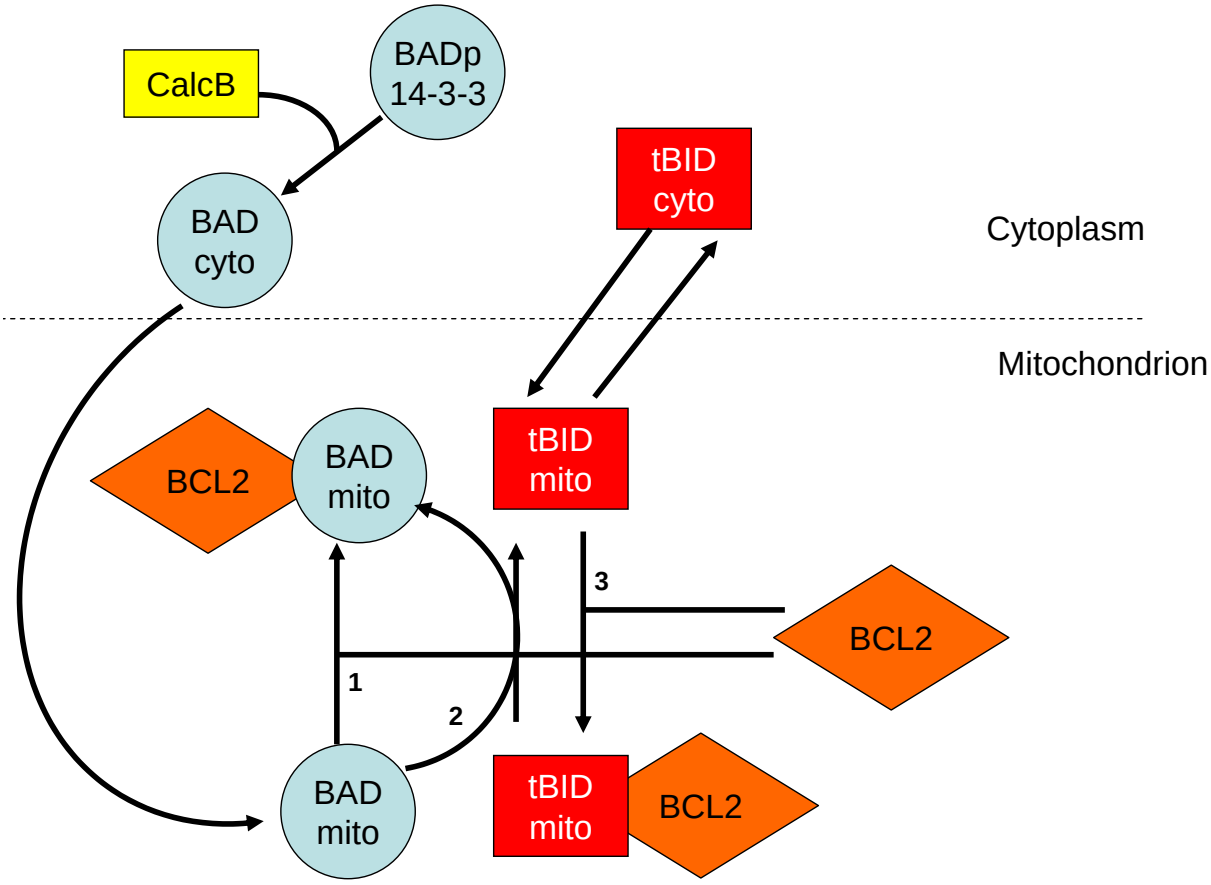


Figure 3.3: Combined Model of BID, BAD, and BCL2 Signaling Pathways.

is similar to the BID model in which the modified multiplier was used. Overflow and underflow protection mechanisms were implemented. Decay and creation were omitted. Competing reaction scenarios also occurred for protein BCL2 and BAD_mito. Similar to the BID model approach,

where a reactant protein is part of two independent reactions, both reactions competed to react with the same reactant protein. Each reaction is capped to only use up a portion of the reactant protein per time unit. For BAD_mito, the two competing reactions 1 and 2 in Figure 3.3 are equivalent in strength. Hence, each reaction is capped to consume at most half of BAD_mito per time unit. Similarly, two competing reactions 2 and 3 in Figure 3.3 for BCL2 are equivalent in strength. Hence, each reaction is capped to consume at most half of BCL2 per time unit.

The following rate constants in Table 3.6 were used in the BAD model. We used the same unit step for time and concentration as well, shown in the third column. Since the modified multiplier was used, the rate constants were adjusted accordingly to compensate for the modified multiplier shift arithmetic. The adjusted rate constant value is shown in the right most column in Table 3.6.

Table 3.6: Reduced BAD Model Rate Constant.

	Rate Constants k_f (unit)		
Reaction	Rate constant value	In discrete system unit step	Adjusted value for shifted multiplier
$\frac{d[cyto]}{dt} = k_f[BAD14][CalcB]$	1.00 $nM^{-1}s^{-1}$	3.33x10^{-3}($\frac{1}{10}nM^{-1}\frac{1}{30}s^{-1}$)	218
$\frac{d[mito]}{dt} = k_f[cyto]$	1.00 s^{-1}	0.0333 ($\frac{1}{30}s^{-1}$)	9
$\frac{d[BADBcl2]}{dt} = k_f[mito][Bcl2]$	1.00 $nM^{-1}s^{-1}$	3.33x10^{-3}($\frac{1}{10}nM^{-1}\frac{1}{30}s^{-1}$)	218
$\frac{d[BADBcl2]}{dt} = k_f[tBcl2][BAD]$	1.00 $nM^{-1}s^{-1}$	3.33x10^{-3}($\frac{1}{10}nM^{-1}\frac{1}{30}s^{-1}$)	218

3.2.1 Complete Model Analysis

Given this combined model of BAD, BID and BCL2, there is only one signaling protein as input, CalcB, compared to the BID model by itself, where both BCL2 and BAD are inputs. Now BCL2 and BAD are part of their own transition system. The model is placed in more constraints and holds different properties. We applied the same analysis approaches as in Figure 3.2. Each protein was subjected to a search for reachable states, and SAT-based induction was performed to

find unreachable states. The initial concentration was picked based on [36] and [40]. In [36], their model suggested that the concentration of all proteins are initially zero except for BCL2 and BID. They had BCL2 and BID initial concentration set at a 1 to 1 ratio. In Table 2 of [40], it specified that the initial concentration for BAD and BCL2 had a 1 to 1 ratio as well. This suggested that BAD, BID and BCL2 should all have a 1 to 1 initial ratio. Our model does not have BID, but instead BID is in the form of tBID_BCL2. We assumed that all BID will become tBID_BCL2 and this complex contains some BCL2. To account for that, the initial concentration of BCL2 should be half of BAD and tBID_BCL2. Hence, if the tBID_BCL2 complex were to break up, we still will maintain a 1 to 1 ratio between BAD, BCL2 and tBID. We chose 25 nM as the initial concentration for BAD, and tBID_BCL2, and 12.5 nM for BCL2. Everything else is set to zero. This is represented in the model as 250 units and 125 units.

First, we analyzed the model by using the GALS tool. In the first run, we were only able to reach 3 unique states for tBID_cyto, which is quite lower than what is expected from a simulation of a random sequence. We observed that when stimulating the system with the sequences that GALS found, the system had not yet reached a steady state. If we just let the system simulates run for one more second of dynamics, which is equivalent to about 30 clock cycles, the system can achieve its steady state concentration and tBID_cyto concentration will rise. This is due to the fact that it takes a longer period of time for the input changes to propagate to tBID_cyto. Similarly, we can reach a few more states for all proteins if we extended the simulation for an additional second. This insures that the system reaches its steady state for all the sequences that GALS found. The approach improved the number of reachable states significantly. Reachability analysis results found the following number of reachable states for each protein out of 256 possible states in Table 3.7. As illustrated by the table, there is significant improvement by using genetic algorithm guided simulation as compared to random simulation. The actual input sequences for tBID_cyto that reached 1.1 nM is recorded in Appendix B.1.2. The parameters used for GALS is summarized in Table 3.3.

Table 3.7: Complete Model Reachability Results.

Protein	Initial	Random	Genetic	Protein	Initial	Random	Genetic
tBID_cyto	0.0 nM	1	**12**	BAD_cyto	0.0 nM	238	**239**
tBID_mito	0.3 nM	15	**103**	BADp_14	25.0 nM	250	**250**
tBID_BCL2	25.0 nM	60	**114**	BAD_mito	0.0 nM	1	**11**
BAD_BCL2	0.0 nM	1	**230**	BCL2	12.5 nM	91	**125**

The SAT-based induction method was applied to the model to prove any unreachable state space. As before, we used strong induction for this approach. For the base case, the circuit was unrolled for 9 time-frames. Due to the complexity of the circuit, if we unrolled anymore time-frames, many instances could potentially time out; hence we chose 9 time-frames as the base case. For the base case, each time-frame was injected with the target state and the same initial concentration as used in GALS. In our first attempt, we partitioned the flip flop by protein type. Any states that cannot be reached by the GALS are injected individually to the SAT-based BMC as illustrated in Figure 2.9 to perform the base case analysis. During the base case run, we were able to reach 3 more states for BAD_BCL2 and 1 more states for BADp_14. If the property was UNSAT, then we performed SAT-based induction. Here, the circuit was unrolled 10 time-frames. The properties were injected according to Figure 2.10, while the initial state was left unconstrained. If this case is UNSAT, we can definitely conclude that this state cannot be reached under any given initial concentration. The Table 3.8 gives a quick summary the analysis result for each protein. R is for reachable states, UR is for unreachable states, and UNK are the number of unknown

Table 3.8: Complete Model Analysis Results.

Protein	R	UR	UNK	Protein	R	UR	UNK
tBID_cyto	12	224	20	BAD_cyto	239	1	16
tBID_mito	103	9	144	BADp_14	251	1	4
tBID_BCL2	114	0	142	BAD_mito	11	0	245
BAD_BCL2	233	0	23	BCL2	125	1	130

states. zChaff was limited to 2400 seconds before timing out. The complete list of the analysis for

tBID_cyto is available in the Appendix B.2.2.

3.2.2 State Space Partition

In addition to partitioning by protein concentration groups, we used a similar partition scheme in the stand alone BID model. Some properties we studied are related to the relationship between proteins in the model. We wanted to know what combinations of BCL2 and tBID_cyto are unreachable from the initial state. Through domain knowledge, we knew that some combinations of high concentration of BCL2 and tBID_cyto were unreachable. Using the SAT-based induction, we constrained the concentration of tBID_cyto between 0.8 and 1.1 nM. Different concentrations of BCL2 were injected into the unrolled circuit to check for satisfiability. We unrolled the circuit for 11 time-frames in the base case and 12 time-frames for the induction case with zChaff time out at 2400 seconds. This method was able to prove that most concentrations of BCL2 that were higher than 5.2 nM will prevent tBID_cyto concentration to rise above 0.8 nM. Some concentrations higher than 5.2 nM were time out by zChaff. However, using SAT-based induction, we were able to prove that at 5.2 nM of BCL2, tBID_cyto cannot rise above 0.8 nM. Hence, we can conclude that any concentration higher than 5.2 nM of BCL2 will have the same effect. These properties helped us study the relationship between BCL2 and tBID_cyto. At any give time, if we measure the concentration of BCL2 and it is equal or greater than 5.2 nM, we can imply that tBID_cyto concentration has to be less than 0.8 nM.

We attempted other partition groups but the results were not as strong. We tried to study the following properties:

- If BCL2 concentration is depleted from 12.5 nM to between 0 and 3.1 nM, can tBID_cyto concentration remain zero? This property studied if BCL2 concentration can decrease without raising tBID_cyto concentration
- Can BAD_BCL2 concentration increase to 19.2 to 22.3 nM while tBID_cyto remains at zero? This property studied if BAD_BCL2 can increase without raising the tBID_cyto concentration

- Can BAD_BCL2 stay a low concentration between 1.0 and 1.5 nM and tBID_cyto concentration raise to 0.8 to 1.1 nM? These properties studied if it is possible to raise tBID_cyto concentration without increasing BAD_BCL2 concentration

When injecting these properties for SAT-based induction, we did not obtain any good results that gave us a clear answer to these questions. Almost all of the injected states were UNSAT for the base case and SAT or timed out for the induction case.

Chapter 4

Fault Model Construction and Analysis

In this chapter, we will discuss how single and double stuck-at fault models were injected to the apoptosis model. A stuck-at fault can be described as a signal stuck at a value, either 1 or 0. This signal remains at the same logic value regardless of what its inputs. Stuck at fault composes up to 50% of all fault occurrences in chips manufacturing. In hardware systems, stuck-at faults can occur when a wire or a gate is either shorted or connected to a source because of manufacture defects or process variations. We would like to study this very simple fundamental fault in an electric circuit to see how it might affect the biological system signaling pathway. This approach introduces an interesting abstraction level by treating faults in biological system as gate level faults. We would like to explore the connection between what a stuck at fault might mean, how it plays a role and how it manipulates the biological pathways. Through the tedious process of mapping the gate level fault back on to the transition diagram, we can manually analyze its effect and its dynamic role on the apoptosis pathways.

During the study, we constructed thousands of fault models and subjected each one to simulation and the results were analyzed. Selected faults were automatically mapped back to its meaning in the transition diagram and interpreted manually. Our objective was to study what type of fault or pair of faults would cause the model to behave differently from our original control model.

By doing this cause-effect analysis, we aimed to find some non trivial faults and any fault relationships that causes the circuit to malfunction and we also explained what each fault implies in a biological system.

4.1 Single Stuck-at Fault Model Construction

Given the complete model as described is Chapter 3, we synthesized the model into a net list in .lev format. The net list has 13,252 gates. Each gate can have two associated independent faults, either a stuck at 1 or a stuck at 0. Hence there are 26,504 faults. It is obvious that a stuck-at fault in the inputs and the outputs will not yield any significant results since inputs are fully controllable and outputs do not propagate to any other gates. Therefore, we chose to omit constructing stuck-at fault models for all inputs and outputs of the system. Any buffer gates were also omitted, since the fault is a redundant fault of its predecessor gate: a fault at a buffer's predecessor is also a fault at the buffer. After a careful study, we chose to omit all NOT gates as well. Any NOT gate that is stuck-at means its complemented predecessor is not stuck-at. Illustrated in Figure 4.1, a NOT gate 101 that has an output signal A that propagates to a set of gates. while its predecessor's gate 100 has an output signal A' propagates to another set of gates. If the NOT gate 101 is stuck at 1, gate

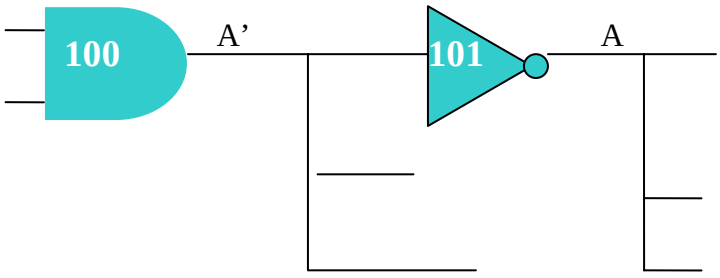


Figure 4.1: Single Stuck at Fault for a NOT Gate.

100 still can change its value freely. For any signal that is being injected with a stuck at fault value, its complement must also be stuck at a complemented value. Having to consider a fault on gate 100 and a fault on gate 101 separately would be redundant. The fault injection for a NOT gate's predecessor will insure a complemented stuck at fault on the NOT gates. Hence, we avoided fault injection for all NOT gates.

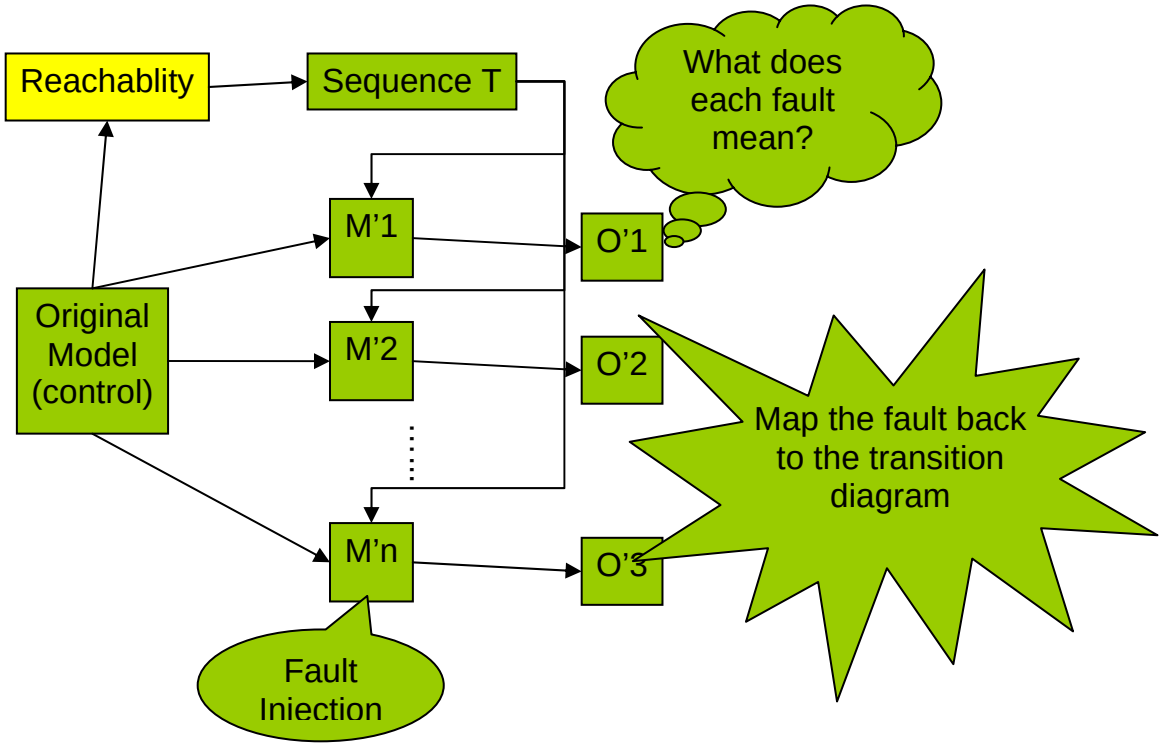


Figure 4.2: Fault Model Construction and Mapping.

After omitting all the above mentioned gates, there are 6910 remaining gates that we needed to consider. Therefore, we have 13,420 different single stuck at fault models. Each of the models were subjected to simulation given two unique input sequences. These sequences brought the original circuit from its initial non-apoptotic state to an interesting target state. The first sequence TA brings the original circuit from its initial state to an apoptotic state, in which the concentration

of tBID_cyto reaches a value of 11 (1.1nM). The second sequence TN brings the original circuit from its initial state to a non-apoptotic state, in which the concentration of tBID_cyto still remains at zero. However, other proteins' concentrations can be different. We also observed, if sequence TA is applied after TN, apoptosis still can occur. The single fault model approach is illustrated in Figure 4.2. Due to the simplicity of the complete model with only one input, we were able to obtain the two input sequences manually, where the inputs are either on or off. The two sequences TA and TN, are either having a lot of CalcB or having very little CalcB, respectively. TA is a sequence of 103 input sets, in which each set is applied at a time-frame. All bits in each input set are one, hence CalcB protein is at its max of 25.5 nM. TN is a sequence of 150 input sets. All bits in the input set are zero, except the last two least significant bits. These two bits maintained the reset signal high and CalcB protein concentration at 0.1 nM.

4.2 Single Stuck at Fault Model Analysis

We first studied the effects of the sequence TA on the 13,420 stuck at fault models. After simulating the circuit with the sequence TA, the final state of the simulation for each model was observed and classified. Sequence TA brought the original circuit from a non-apoptotic state, where the tBID_cyto concentration is zero, to an apoptotic state, where the tBID_cyto concentration is above 1.1 nM. There were 10,153 fault models that still became apoptotic; of which are the faults that did not cause the circuit to malfunction. However, there were 511 faults that kept the circuit at a non-apoptotic state. The remaining 2,756 faults brought the circuit to other less distinct state where the tBID_cyto concentration was between 0.1 and 1.0 nM or if it is greater than 1.1 nM. We focused our study here on the 511 faults, which caused the system to malfunction.

By using equivalent fault collapsing tool, we were able to group the 511 faults into 274 equivalent sets of non-redundant faults. Each set of faults was unique and disjoint. In the next step, we will discuss how to map each of these sets of faults back to the signal in the transition

diagram.

4.2.1 Fault Mapping

Given a faulty gate, our objective was to map that faulty gate back to the wire or register representation in the Verilog code. However, due to the conversion steps, only partial mappings were retained during the transformation process. Primary inputs, primary outputs and registers were the only signals that can be mapped back. Due to this limitation, we modified the Verilog model to have more outputs which made more signals observable and traceable. This assisted us in mapping more faults and analyzed the effect of the faults. Refer to Figure 2.3 for the transformation of the circuit from Verilog to gate level .lev. A full mapping was retained when transforming from .bench to .lev. However, most mapping was lost during the transformation from structural Verilog to blif and from blif to bench. Only partial mapping of inputs, outputs and registers were retained. We developed a small program to automate the mapping process, called Mapping Automated Program (MAP). MAP generated a text file that contains a mapping between each gate and the corresponding Verilog signal. MAP was also used to group a list of single faults into equivalent sets of faults. Instruction on how to use MAP is included in the Appendix A.3.

Given an equivalent set of faults generated by MAP, if a fault in an equivalent set is mappable, then it is sufficient to conclude that the whole set is mappable since they are all equivalent. MAP also helped us with this mapping process by generating mapping files. This mapping file contained a list of mappable fault for each equivalent set along with its wire name in Verilog. From the 274 equivalent sets above, 93 sets are mapped. The number of faults that can be mapped is currently limited due to the lost of signals tracking during the conversion process Figure 2.3. Mapping can be improved in two ways: either by placing more output signals to make more signals observable or finding an alternative conversion process that retains more mapping information. Of those 93 mapped sets, 76 sets contained at least one mappable fault that is an output signal of a

gate. Other 17 sets, individually had only mappable input fault signals without any mappable output fault. Without the output fault mapped, we would not be able to interpret the meaning of this fault. Hence, we are only interested in looking at the 76 sets with output faults.

Similarly, we studied the sequence TN on 13,420 stuck at fault models. TN kept the original circuit in a non-apoptotic state. There are 11,616 faults that remained at a non-apoptotic state of which tBID_cyto concentration remained at zero. Recall that an apoptotic state means tBID_cyto concentration reaches a value of 11 or 1.1 nM. The remaining 1,804 faults caused the tBID_cyto concentration to rise above 0. Out of those 1,804, there were 890 that caused the tBID_cyto concentration to rise above 10 or 1.0 nM, which is an apoptotic level. The other 914 faults led tBID_cyto concentration to a value between 0.1 and 0.9 nM. We used the MAP tool to group 890 faults to 457 equivalent sets. Out of the 457 sets, MAP was able to map 124 sets, in which 106 sets had at least one output fault. Table 4.1 summarizes all the statistics we mentioned in the previous page.

Table 4.1: Single Stuck at Fault Results.

	TA			TN		
# Model	13,420			13,420		
	faulty	not faulty	semi-faulty	faulty	not faulty	semi-faulty
# Model	511	10,153	2,756	890	11,616	914
# unique set	273	7,146	NA	457	8,124	NA
# mappable set	93	457	NA	124	718	NA
# set w/ output	**76**	251	NA	**106**	443	NA
% mapped	27.7	3.5	NA	23.2	5.5	NA

Row 4 classified all fault models in three groups based on its simulation result as either faulty, none faulty or semi-faulty. For sequence TA, the semi-faulty were models with tBID_cyto concentration between 0.1 and 1.0 nM or above 1.1 nM. For sequence TN, the semi-faulty are models with tBID_cyto concentration between 0.1 and 0.9 nM. Row 5 is the number of equivalent sets of faults for each classification. Row 6 shows how many of those sets were mapped. Row 7 displays the number of mapped sets with at least one output fault that we will conduct our study.

and Last row 8 is the percentage of fault that we conducted our study. By observing percentage mapped for faulty and none faulty models, we can see that the ratio is much higher for the faulty models. Also note that for a fault to be mappable, its signal must be either a register or connected to an output. Since most outputs are connected to different signals involved with differential equations' arithmetic such as multiplication and addition of protein concentrations, this bias shows that the faults that are related to the those arithmetic area and registers are more likely to cause the system to malfunction. We will present the results and the interpretations of a few mapped faults for each sequence in the following subsections.

4.2.2 Results for Sequence TA

From the 76 unique mapped faults, there are 15 functionally distinct faults. Faults that are considered not distinct are the ones that are located on the same structural bus. The full, complete list of faults and a short interpretive description is available in the Appendix B.3.1. Here, we will discuss in detail three less trivial faults illustrated in Figure 4.3.

Recall that the sequence TA brings the original circuit from a non-apoptotic state to an apoptotic state in the original healthy system. Each of these 76 faults here prevented the system from becoming apoptotic. The first faulty system had the most significant bit on the reaction with CalcB stuck at 0, which meant that half of the BADp14 in the reaction leaks before arriving to BAD_cyto. This kept BAD_cyto concentration lower, hence BAD_mito lower and tBID_mito lower. The second faulty system had the fault on the translocation of BAD from the cytoplasm to the mitochondria. The fault was quite severe since it is the middle bit. The third faulty system kept the BCL2 concentration at a higher level. The biological system interpretation of this faulty system is not yet clear. This is a great finding to help us pinpoint a particular subsection of the diagram for further experimental studies.

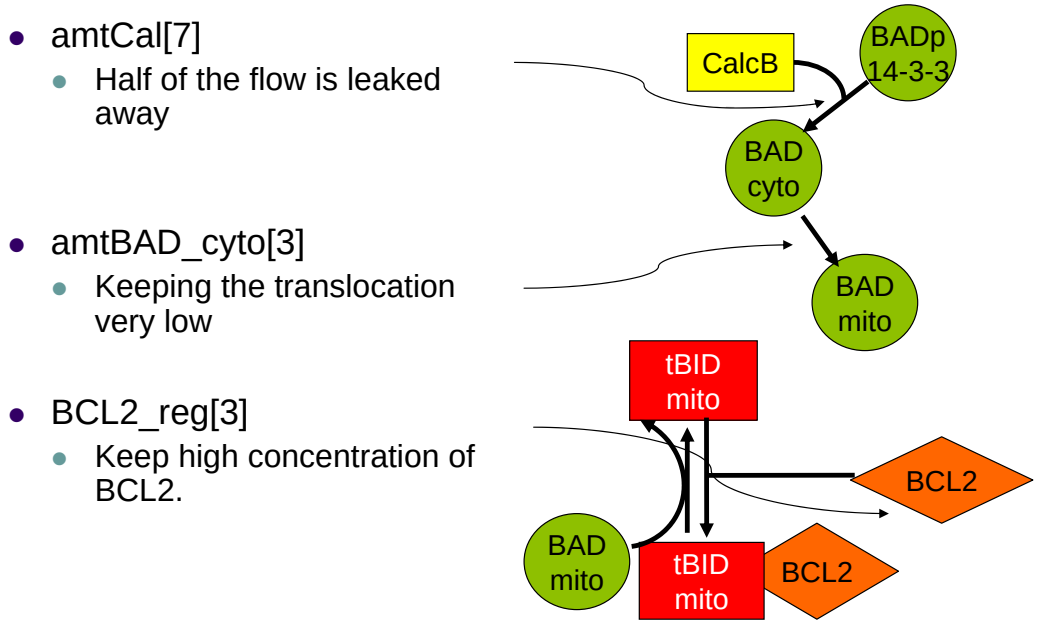


Figure 4.3: Single Stuck at Fault Results for Sequence TA.

4.2.3 Results for Sequence TN

From the 107 unique mapped faults, they described 13 functionally distinct faults. This means that the faults are located in a different parts of the system, as compared to two faults located on the same structural bus. We will discuss and interpret the meaning of the following three faults as illustrated in Figure 4.4. The full list of faults is available in the Appendix B.3.2.

The first fault caused the translocation of BAD_cyto to be stuck at 1. The translocation of BAD is a 8-bit wide bus; having bit #1 stuck at 1 created a slight constant flow of BAD to the mitochondria. By raising the concentration of BAD_mito, BAD_mito can break up more tBID_BCL2 complexes and free tBID, which in turn translocates itself to the cytoplasm and causes apoptosis. The second fault directly added more BAD_mito complex by having the register bit stuck at 1. This fault caused a similar effect as the first fault. The third fault created a depletion of BCL2 complex, which caused the model to become apoptotic. The biological interpretation has

not yet determined. However, by observing the flow diagram, depleting BCL2 causes both reactions that form tBID_BCL2 complex and BAD_BCL2 complex to slow down, hence leaving more BAD_mito to break up tBID_BCL2 complex and more tBID_mito.

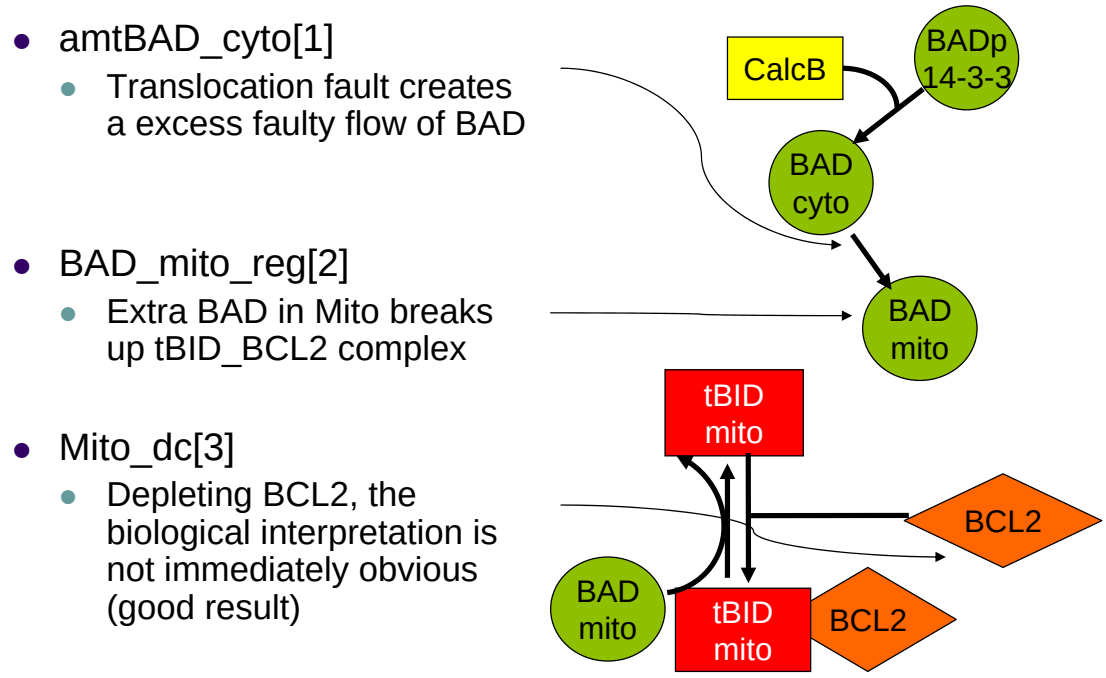


Figure 4.4: Single Stuck at Fault Results for Sequence TN.

Many of the single stuck-at faults are quite obvious in their interpretation, such as faults that increase tBID_mito or tBID_cyto directly will cause apoptosis. However, those faults are important to observe in order to help us to validate our model and understand its properties. For those less trivial faults. They are very interesting faults that pinpoint the weakness of the system where a single manipulation can cause the entire system to malfunction. Further experimental study can be carried out using the faults analysis to form hypotheses. Furthermore, we conducted studies on double stuck-at fault models in the next section.

4.3 Double Stuck at Fault Model Construction and Analysis

Given a set of single-stuck at faults that did *not* cause the system to malfunction, we paired them up and injected them simultaneously into the original circuit to construct double fault models. Recall from the previous single fault study, we had two sets of single faults; one set that used sequence TA and the other set that used sequence TN. For sequence TA, there are 10,153 faults that did not cause the system to malfunction, which meant that it still led the system to reach an apoptotic state. Of the 10,153, there were 7,146 equivalent sets of faults and 251 faults were mapped. Using the 251 mapped faults, we paired up all pair combinations of faults and constructed double fault models. There are 31,320 different double fault models. Of those, there are 29,753 double faults that did not cause the circuit to malfunction; the other 1565 pairs of faults caused the tBID_cyto concentration to reach between 0.1 and 1.0 nM or above 1.1 nM. Only 3 pairs of faults caused the circuit to become non-apoptotic. Table 4.3 shows the number of faults and the corresponding mapping to its Verilog wire name.

Similarly, double fault models construction was performed for TN sequence. There are 443 single faults that were mapped, hence giving us 97,783 double fault models. Out of those, only 15 pairs of faults caused the tBID_cyto concentration to rise above 1.0 nM, which is an apoptotic state. The others 59 pairs caused tBID_cyto concentration to reach between 0.1 and 0.9 nM. The majority remaining pairs do not lead the system out of non-apoptotic state. Table 4.2 and Table 4.1 quickly summarizes all the statistics mentioned above.

Table 4.2: Double Stuck at Fault Statistics.

	TA			TN		
Single Fault	faulty	not faulty	semi-faulty	faulty	not faulty	semi-faulty
# mapped	76	**251**	NA	106	**443**	NA
# Model	31,320			97,783		
Double Faults	faulty	not faulty	semi-faulty	faulty	not faulty	semi-faulty
# Model	**3**	29,752	1,565	**15**	97,709	59

4.3.1 Results for Sequence TA

From Table 4.3, all three faults have a common single fault of *5748*. In fact, the first and the third fault are exactly the same. Fault *50* and fault *13203* are both pointing to the same register tBID_mito bit 7. If the gate value is negative, that means the gate is a stuck-at zero fault. Recall that sequence TA led the system to an apoptotic state, however with the presence of these faults, the faulty system remained at non-apoptotic state.

Table 4.3: Double Stuck at Fault Results for Sequence TA.

-50	-5748	tBID_mito_reg[7]	NOT_amtCyto[1]
12075	-5748	NOT_tMito_addv[7]	NOT_amtCyto[1]
-13203	-5748	tBID_mito_reg[7]_in	NOT_amtCyto[1]

For the first pair of faults, tBID_mito register bit 7 was stuck at 0. By interpreting the transition diagram, we observed that if the tBID_mito concentration is lower, then the tBID_cyto would be lower as well. However, this single fault alone did not cause any erroneous behavior. Through simulation, we observed that bit 7 in tBID_mito was not even used. The NOT_amtCyto fault was also stuck at zero, which meant that signal amtCyto was stuck at 1. This caused an excess amount of BID_mito and raised the tBID_cyto concentration up to 12 or 1.2 NM at some point. However, when combined, tBID_mito concentration did bring the tBID_cyto concentration up at apoptotic state at some point. However, since tBID_mito bit 7 was stuck at zero, it caused a failure in our overflow protection component. Hence, the tBID_mito concentration rolled over to zero and caused the tBID_cyto to drop back to zero. Similarly, the second and third pairs showed similar behavior. These three pairs of faults demonstrated the possibility that the faults can cause failures in the model that would be impossible to achieve in a biological system.

4.3.2 Results for Sequence TN

From Table 4.2, there are 15 pairs of faults that caused the system to become apoptotic. Of those 15 pair of faults, there were only 5 sets of faults that were located in unique locations in the transition diagram. The complete sets of double faults is available in the Appendix B.3.4. Table 4.4 lists the five sets of faults' wire name along with its stuck at value.

Table 4.4: Double Stuck at Fault Results for Sequence TN.

1	BAD_cyto_reg[2]	1	NOT_amtBAD_cyto[0]	1
2	NOT_aBCL2[3-7]	0	NOT_BCL2_inc[3-7]	1
3	NOT_atBCL2[3-7]	0	NOT_tMito_dc[3-7]	1
4	NOT_atBCL2[0]	1	NOT_tMito_dc[1]	1
5	NOT_wiretBCL2[0]	1	NOT_tMito_dc[1]	1

For faults 2 and 3, each were composed of five separate pairs of faults with each pair having the same significant bit. Such as signal NOT_aBCL2[3] is paired up with signal NOT_BCL2_inc[3], and so on. We will discuss the effects of a pair NOT_aBCL2[3] and NOT_BCL2_inc[3] as an example for the rest of the fault in set 2. Signal NOT_aBCL2[3] is stuck at zero, which meant that signal aBCL2[3] is stuck at 1. This caused an excessive reaction between BAD_mito and BCL2, which created BAD_BCL2 complex. However, by itself, the fault depleted the concentration of BAD_mito to break of tBID_BCL2 complex, hence no apoptosis occurred. We also noticed that signal aBCL2 is one component that makes up the signal BCL2_inc, that signal aBCL2 determines how much is depleted from BCL2, and signal BCL_inc determines how much is depleted from BAD_mito. Signal BCL_inc is a summation of signal aBCL2 and signal atBID. When signal NOT_BCL2_inc[3] is stuck at 1, it meant that signal BCL2_inc[3] is stuck at zero. In a normal condition, signal BCL2_inc[3] bit is always at zero, so this fault by itself had no effect on the system. When combining those two faults, signal aBCL2[3] still accelerated the reaction of BAD_mito with BCL2 depleting both proteins. However, since signal BCL2_inc[3] was stuck at zero, the effect of signal aBCL2 depleting BAD_mito was blocked. The overall effect was that

the BCL2 concentration was depleted more through a leak without reacting with the BAD_mito. Hence, it left more of the BAD_mito to break up the tBID_BCL2 complex and caused apoptosis.

Similarly, the third set has signal atBCL2 as a component of signal tMito_dc that acted on the reaction of tBID_mito with BCL2. Having signal atBCL2 stuck at 1, it accelerated the reaction of tBID_mito with BCL2 to form tBID_BCL2 complex. However, signal tMito_dc stuck at 0 blocked the depletion of tBID_mito while BCL2 was still depleting. Hence, it left more tBID_mito to cause apoptosis. The overall effect was that BCL2 was depleted more quickly by the reaction of tBID_mito and BCL2.

Sets 4 and 5 have the same behaviors and effects, where signal wiretBCL2 is directly related to signal atBCL2. Here, signal atBCL2[0] is stuck at 0, it slowed down the reaction of tBID_mito with BCL2 by only allowing an even unit of protein to interact. This kept both concentrations of tBID_mito and BCL2 higher and prolonged the depletion of BCL2. With the signal tMito_dc[1] stuck at zero fault, it blocked the depletion of tBID_mito even more. The overall effect was that BCL2 was depleting twice as fast as tBID_mito in the reaction.

Similarly, all three set of faults, 2, 3, 4 and 5 have the similar overall effect. The concentration of BCL2 was depleted faster than in a normal condition. This depletion was due to a leakage or over consumption of BCL2 for a particular reaction. It is not yet obvious how the lack of BCL2 caused the system to become apoptotic. Previous study in [36], has shown that BCL2 prevents apoptosis. Our result further confirmed this relationship and its effects can be further studied to observe how BCL2 causes the system to malfunction.

The first set of faults has the most interesting behaviors that is not intuitive by observing the signaling pathways. We have not yet formulated any straight-forward biological interpretations or explanations for why such phenomenon occurred. However, we attempted to interpret its behaviors based on our transition model. A subsection of the model is illustrated in Figure 4.5. The BAD_cyto register bit 2 was stuck at 1, which meant that there was a constant faulty creation of BAD_cyto that kept its concentration level at bit 2 always high. By itself, this fault created an

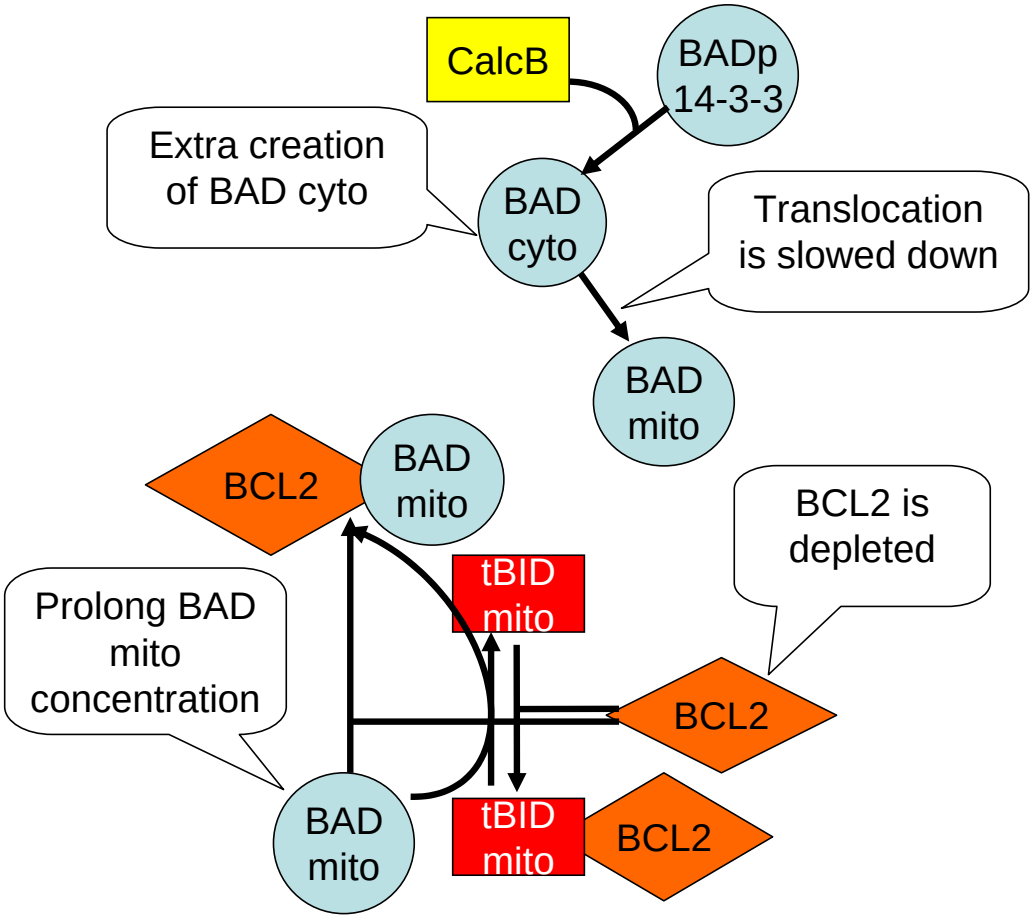


Figure 4.5: Double Stuck at Fault for Sequence TN.

excess amount of BAD_mito that broke up more tBID_BCL2 complex and caused tBID_cyto to rise. However because the tBID_mito concentration was not high enough to become apoptotic, the tBID_cyto concentration still remained at zero. For signal amtBAD_cyto fault stuck at 0, this fault slowed down the translocation of BAD_cyto to BAD_mito by a just a little since it is at bit 0. This fault by itself caused BAD_cyto to accumulate and prolonged the translocation. There was less BAD_mito and tBID_mito concentration was much lower than in the normal condition. When these two faults were combined, there was an extra creation of BAD_cyto and yet the transloca-

tion was slowed down. The BAD_cyto concentration was even higher than just slowing down the translocation alone. Because BAD_cyto concentration was kept so high, the translocation of BAD became more steady and stronger for a longer period of time. This prolonged, steady flow of BAD helped deplete BCL2. After that, the tBID_mito concentration rose and caused apoptosis.

A biological system interpretation for what this might mean is not yet apparent. This is a good result to help us to pinpoint a subsection of the signal pathway that needs further study and experimentation. Based on this observation, we can form hypotheses in which laboratory experiments can be conducted to confirm or reject the hypotheses.

Chapter 5

Conclusion

We have introduced a unique level of abstraction by studying the apoptosis signaling pathways as a finite transition system. This approach allows us to extract the system properties and behaviors through utilizing hardware testing and verification techniques. These techniques assist us in studying the realizability of protein concentrations without fully relying on simulation. Such information can be extremely valuable to the understanding of apoptosis. Furthermore, we can study cause-effect fault analysis through constructing single and double stuck at fault models.

5.1 Summary

We have presented the construction of a stand alone BID model and the combined model for a section of an apoptosis signaling pathway. We built a finite transition system using Hardware Description Language Verilog. The transition system used protein concentration to define its state space. External signaling proteins were treated as inputs and ordinary differential equations were used to define the state transition function. Using previous experimental literatures, we chose representative values for the reaction rate constants. Slower reactions dynamics, such as creation and decay, were omitted because of the short time scope of our study. Academic and industrial

tools were used to synthesize and convert the model to numerous formats.

Two orthogonal verification approaches were applied to analyze the models. Genetic Algorithm (GA) guided Logic Simulation (GALS) utilized genetic algorithm to search for an input sequence that can bring the system from an initial state to a target state. A* search with Hamming distance heuristic used the guide from GA to perform a search from a specified initial state to a target state. The hybrid approach used both random logic simulation and genetic algorithm to guide the generation of input sequence for the search process. GALS was very effective in finding a large set of states in the reachable state space as compared to formal verification methods. GALS avoided both time and memory explosion problems that formal verification methods may potentially face. Because GALS is incomplete, we used SAT-based verification methods to further explore the reachable missed by GALS and provide proofs for unreachable state spaces. SAT-based induction with depth was used, in which the base case employs SAT-based bounded model checking (BMC) methods, to prove unreachable state spaces. SAT-based BMC was able to further find more reachable states in some instances. For those that are unreachable, we used SAT-based induction to find a proof that it is unreachable. Furthermore, different flip-flop partition schemes were also used to extract more interesting properties and relationships between proteins.

After studying the control model, we performed manipulations to the model by introducing faults into the system and studied the cause and effect of such manipulations. Single fault models were first constructed by injecting a single stuck-at fault to the control model. Simulation was used to distinguish between faulty models that have erroneous responses and those that have no erroneous responses. For those faulty models, we performed fault mapping to trace the faults back to their locations on the transition diagram. We attempted to draw explanations and interpretations for the causes of those faults and form hypotheses for laboratory experiments. For those single fault models that did not have erroneous responses, we paired them up and injected them simultaneously to construct double-fault models. Similarly, simulation was used to distinguish between malfunction and non-malfunction double fault models. Pairs of faults, in which by themselves

did not cause the circuit to malfunction, that produced erroneous responses had shown interesting behaviors that required further study for an explanation.

5.2 Future Work

In future work, a more extensive and refined model can be constructed. With more experimentation and continuous model analysis results, we can obtain more reaction rate constants and refine our model further to include more reaction pathways. With a few proteins removed from the BID and BAD model, we can include them back for the next iteration of the model. Furthermore, we aim to incorporate slower dynamic reactions and other pathways in our model. In parallel, improved and new verification approaches can be developed with the current framework. Hence, we can extract more properties of the system and classify more states to be reachable or unreachable.

For the single fault model, there was only about 25% of the faults that were mapped back to the transition diagram. This was due to the loss of mapping information during the conversion process from high level Verilog to low level net list. We can find alternative conversion methods that keep more mapping information or find better mapping approaches to be able to map more faults. Further, we can conduct effect-cause analysis, as in contrast to cause-effect analysis that was done in this work. Effect-cause analysis essentially comes down to a hardware diagnostic problem. Given an ill behavior, we identify a set or sets of faults that can cause such behaviors. From a biologists' view, this can help pinpoint the causes for known ill behaviors. It helps to answer the questions of what can cause the system to malfunction in such specified ways. This can further be extended in any application to a specific disease or illness. Our approach holds great potential for significant findings by using this type of abstraction.

Bibliography

[1] A. Csikasz-Nagy, D. Battogtokh, K.C. Chen, B. Novak, and J.J. Tyson. Analysis of a generic model of eukaryotic cell-cycle regulation. *Biophysical Journal*, 90(12):4361 – 79, 2006/06/15.

[2] Pierre Golstein. Cell Death in Us and Others. *Science*, 281(5381):1283–, 1998.

[3] Linda J. Miller and Jean Marx. Apoptosis. *Science*, 281(5381):1301–, 1998.

[4] P. Bernardi, V. Petronilli, F. Di Lisa, and M. Forte. A mitochondrial perspective on cell death. *Trends In Biochemical Sciences*, 26(2):112–117, 2001.

[5] P. Bernardi, L. Scorrano, R. Colonna, V. Petronilli, and F. Di Lisa. Mitochondria and cell death - mechanistic aspects and methodological issues. *European Journal Of Biochemistry*, 264(3):687–701, 1999.

[6] M. D. Esposti. Mitochondria in apoptosis: past, present and future. *Biochemical Society Transactions*, 32:493–495, 2004.

[7] J. C. Goldstein, C. Munoz-Pinedo, J. E. Ricci, S. R. Adams, A. Kelekar, M. Schuler, R. Y. Tsien, and D. R. Green. Cytochrome c is released in a single step during apoptosis. *Cell Death And Differentiation*, 12(5):453–462, 2005.

[8] D. R. Green and J. C. Reed. Mitochondria and apoptosis. *Science*, 281(5381):1309–1312, 1998.

[9] B. Mayer and R. Oberbauer. Mitochondrial regulation of apoptosis. *News In Physiological Sciences*, 18:89–94, 2003.

[10] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome: a knowledgebase of biological pathways. *Nucleic Acids Research*, 33:D428–D432, 2005.

[11] J. R. Burch, E. M. Clarke, K. L. McMillan, and David L. Dill. Sequential circuit verification using symbolic model checking. In *DAC '90: Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 46–51, New York, NY, USA, 1990. ACM Press.

[12] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 365–373, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[13] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 317–320, New York, NY, USA, 1999. ACM Press.

[14] Parosh Aziz Abdulla, Per Bjesse, and Niklas E & #233;n. Symbolic reachability analysis based on sat-solvers. In *TACAS '00: Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 411–425, London, UK, 2000. Springer-Verlag.

[15] A. Gupta, Yang Zijiang, P. Ashar, and A. Gupta. Sat-based image computation with application in reachability analysis. Formal Methods in Computer-Aided Design. Third International

Conference, FMCAD 2000. Proceedings (Lecture Notes in Computer Science Vol.1954), pages 354–71, Austin, TX, USA, 2000. Springer-Verlag.

[16] D. Spierings, G. McStay, M. Saleh, C. Bender, J. Chipuk, U. Maurer, and D. R. Green. Connected to death: The (unexpurgated) mitochondrial pathway of apoptosis. *Science*, 310(5745):66–67, 2005.

[17] K. A. Janes, J. G. Albeck, S. Gaudet, P. K. Sorger, D. A. Lauffenburger, and M. B. Yaffe. Systems model of signaling identifies a molecular basis set for cytokine-induced apoptosis. *Science*, 310(5754):1646–1653, 2005.

[18] T. Bouwmeester, A. Bauch, H. Ruffner, P. O. Angrand, G. Bergamini, K. Croughton, C. Cruciat, D. Eberhard, J. Gagneur, S. Ghidelli, C. Hopf, B. Huhse, R. Mangano, A. M. Michon, M. Schirle, J. Schlegl, M. Schwab, M. A. Stein, A. Bauer, G. Casari, G. Drewes, A. C. Gavin, D. B. Jackson, G. Joberty, G. Neubauer, J. Rick, B. Kuster, and G. Superti-Furga. A physical and functional map of the human tnf-alpha nf-kappa b signal transduction pathway. *Nature Cell Biology*, 6(2):97–+, 2004.

[19] J. M. Adams. Ways of dying: multiple pathways to apoptosis. *Genes & Development*, 17(20):2481–2495, 2003.

[20] K. H. Cho, S. Y. Shin, W. Kolch, and O. Wolkenhauer. Experimental design in systems biology, based on parameter sensitivity analysis using a monte carlo method: A case study for the tnf alpha-mediated nf-kappa b signal transduction pathway. *Simulation-Transactions Of The Society For Modeling And Simulation International*, 79(12):726–739, 2003.

[21] M. D. Esposti. The roles of bid. *Apoptosis*, 7(5):433–440, 2002.

[22] B. Antonsson. Mitochondria and the bcl-2 family proteins in apoptosis signaling pathways. *Molecular And Cellular Biochemistry*, 256(1-2):141–155, 2004.

[23] S. Cory and J. M. Adams. The bcl2 family: Regulators of the cellular life-or-death switch. *Nature Reviews Cancer*, 2(9):647–656, 2002.

[24] M. J. Goldenthal and J. Marin-Garcia. Mitochondrial signaling pathways: A receiver/integrator organelle. *Molecular And Cellular Biochemistry*, 262(1-2):1–16, 2004.

[25] T. Dohi and D. C. Altieri. Mitochondrial dynamics of survivin and "four dimensional" control of tumor cell apoptosis. *Cell Cycle*, 4(1):21–23, 2005.

[26] Michael H. Cardone Cynthia L. Stokes Fei Hua, Melanie G. Cornejo and Douglas A. Lauffenburger. Effects of bcl-2 levels on fas signaling-induced caspase-3 activation: Molecular genetic tests of computational model predictions. *The Journal of Immunology*, (175):985–995, 2005.

[27] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 530–535, New York, NY, USA, 2001. ACM Press.

[28] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[29] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley, 1989.

[30] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann. Sequential circuit test generation in a genetic algorithm framework. 31st Design Automation Conference. 31st DAC Proceedings 1994 (IEEE Cat. No.94CH3408-2), pages 698–704, San Diego, CA, USA, 1994. ACM.

[31] F. Corno, P. Prinetto, M. Rebaudengo, M. S. Reorda, and M. Violante. Exploiting logic simulation to improve simulation-based sequential atpg. Proceedings. Sixth Asian Test Sym-

posium (ATS'97) (Cat. No.97TB100205), pages 68–73, Akita, Japan, 1997. IEEE Comput. Soc.

[32] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. *Theory and Applications of Satisfiability Testing. 6th International Conference, SAT 2003. Selected Revised Papers (Lecture Notes in Comput. Sci. Vol.2919)*, pages 341 – 55, 2004//.

[33] S. Subbarayan and D.K. Pradhan. Niver: non-increasing variable elimination resolution for preprocessing sat instances. *Theory and Applications of Satisfiability Testing. 7th International Conference, SAT 2004. Revised Selected Papers (Lecture Notes in Computer Science Vol.3542)*, pages 276 – 91, 2005//.

[34] Mary Sheeran, Satnam Singh, and Gunnar St & #229;lmarck. Checking safety properties using induction and a sat-solver. In *FMCAD '00: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, London, UK, 2000. Springer-Verlag.

[35] P. Bjesse and K. Claessen. Sat-based verification without state space traversal. *Formal Methods in Computer-Aided Design. Third International Conference, FMCAD 2000. Proceedings (Lecture Notes in Computer Science Vol.1954)*, pages 372 – 89, 2000//.

[36] E. Z. Bagci, Y. Vodovotz, T. R. Billiar, G. B. Ermentrout, and I. Bahar. Bistability in apoptosis: roles of bax, bcl-2, and mitochondrial permeability transition pores. *Biophysical Journal*, 90(5):1546–59, 2006.

[37] Sheng Shuo and M. S. Hsiao. Efficient sequential test generation based on logic simulation. *IEEE Design & Test of Computers*, 19(5):56–64, 2002.

[38] Qingwei Wu and Michael S. Hsiao. Efficient sequential atpg based on partitioned finite-state-machine traversal. *itc*, 00:281, 2003.

[39] Qingwei Wu and Michael S. Hsiao. A new simulation-based property checking algorithm based on partitioned alternative search space traversal. *IEEE Trans. Comput.*, 55(11):1325–1334, 2006.

[40] Marie-Veronique Clement David Hsu P.S. Thiagarajan Geoffrey Koh, Huey Fern Carol Teong. A decompositional approach to parameter estimation in pathway modeling: A case study of the akt and mapk pathways and their crosstalk. *Bioinformatics*, 00(00):1–10, 2006.

Appendix A

Instructions

A.1 Conversion Instructions

Synopsis tool SETUP .synopsys_dc.set file should have the following

set link_library * lsi_10k.db
set target_library lsi_10k.db
set synthetic_library ""
set link_library * lsi_10k.db
set symbol_library lsi_10k.sdb the library are located at

set search_path [list . $synopsys_root/libraries/syn]
synopsys_root: /software/synopsys/2003.06/

The following steps are how to convert a Behavioral verilog into series of different format: structural verilog, blif, bench and lev.

Synthesize design using Synopsis: (behavorial verilog => structural verilog)

//start dc_shell with an input script to ignore complex gate
`http://filebox.vt.edu/users/lamhuy/ignore_complex_gates.script`
dc_shell –f ignore_gate_script.script
//if just one top module
//read in a module
dc_shell »read -format verilog sample_rtl.v
//else if more than more module
//analyze each verilog module
dc_shell»analyze –format verilog subModule.v
dc_shell» analyze –format verilog topModule.v
dc_shell »elaborate topModule

//synthesize the elaborated topModule
dc_shell »uniquify -force
dc_shell »compile -map_effort medium
dc_shell »ungroup -all -flatten
dc_shell »write -format verilog -hierarchy -output sample_s.v
use check_design to check warning message
Now *sampe_s.v* is a structural verilog file

Rename structure verilog gate name to work for ABC
//run this executable under DOS command prompt. Make sure sample_s.v is in the same //directory as *LSI_10K_TO_ABC.exe*
`http://filebox.vt.edu/users/lamhuy/LSI_10K_TO_ABC.exe`
LSI_10K_TO_ABC.exe sample_s
//output file will be sample_s_f.v
Change any DFF ordering here

Current ABC tools are available at:
http://`http://www.eecs.berkeley.edu/~alanmi/abc/`

Run ABC to get blif (structural verilog to blif)
//run the 60818 release under DOS command prompt to get blif
`http://filebox.vt.edu/users/lamhuy/abc/abc60818.exe`
»abc60818.exe
abc 01> read_verilog sample_s_f.v
abc 02> strash
abc 03> short_names *//rename all wire**
abc 03> write_blif sample.blif
//manually remove clk or reset signal in input if applicable

Run ABC to get bench (structural verilog to bench)
//run the 61210 under DOS command prompt to get bench
`http://filebox.vt.edu/users/lamhuy/abc/abc61210.exe`
»abc61210.exe
abc 01> read_verilog sample_s_f.v
abc 02> strash
abc 03> short_names *//rename all wire**
abc 03> write_bench sample.blif
//manually remove clk or reset signal in input if applicable

Run ABC to get bench (blif to bench)
//manually remove clk or reset signal in input if applicable
//run the 61210 under DOS command prompt to get bench

»abc61210.exe
abc 01> read_blif sample.blif
abc 02> strash
abc 03> short_names *//rename all wire**
abc 03> write_bench sample.blif
**rename all wire using generic name (optional). However it is necessary if the blif file generated is needed to be read in by VIS tool for analysis later. VIS tool has a bug reading in wire name with _ and /. By renaming all wire, that avoid the bug.*

Run level.dat to get .lev (bench to lev)
//find and replace BUFF in bench generated by ABC to BUF
//save file as UNIX encoding instead of DOS
//find an replace
VDD = OR (a, not(a))
VSS (or GND) = AND (a, not(a))
»./level.dat sample
This creates a .lev and .name for mapping

A.2 Verification Instructions

A.2.1 Reachability Analysis

To run the executable:
./a.exe Apoptosis reach
This command will require:
Apoptosis.lev: circuit in netlist format
Apoptosis.init: initialization file
Apoptosis.target: target state to be searched

Init file format:
Row 1: specify the number of flip flop in the circuit
Row 2: specify the initial state of the flip flop
Row 3: 1/0 specify that this input is fully controllable, X specifies that this input will be fixed at a constant value specify later.

NOTE READ THIS: For the apoptosis circuit, a reset input pin was added in the .bench format as active low. To accommodate this, when compile, -D RESET option should be turn on. This option allows the assumption that the last pin is an active low reset pin. That the reset pin should always be kept high, hence the program will keep the last input as high when compile with that option. So in the init file, the last input should always be 1. For example, if the circuit has 8 input + 1 reset input, in the init file, you only have to specify the condition of the 8 inputs with 111111111.

Last input does not need to be specified. The program assumes it is a reset pin input. Hence this compiled executable can not be used for other circuit unless recompiled is done without -D RESET
Row 4: specify the resolution of input, flip flop. This is the resolution of each protein concentration. The number here is used to help converting binary to decimal output.
Row 5: specify how many set of inputs are fixed as a constant. Set = #Xs in row 3/row4
For most case, if RATE constants are not part of the input, this should be zero. (00, must be in double zero) and Row 3 has no Xs.
Row 6 . . . : each row here specifies the rate constant value to keep the input at. If row 5 has value of 03, then row 6, 7, 8 specify the rate constant.
Row "9": specify how many set of target is the program going to be searching for. Each set of target starts with specifying the number of flip flop in the circuit, ends with keyword: "END". They must be all in the .target file back to back

Target file format:
Contain list of target states specified by 1, 0, X. Each set starts with specifying the number of flip flop in the circuit, ends with keyword: "END"
NOTE: for each set of target, for best search result, they all should have the same "don't care" bit or Xs in the same column.

Program execution:
After running command ./a.exe Apoptosis, the program read in the circuit and initialize it using .init file. Then inquires the user:
Reachabilty search or quit? (reach, quit)?

Original Target (Y/N)?
Yes: ignore all "don't care" target flip flop, only consider non don't care state space
No: consider all flip flop state space
Example: if the target state specified is "1X0"
Yes: ignore middle bit, so the state space would be just 10, 11, 00, 01
No: fully consider all flip flop having the state space of 000, 001, 010, 011, 100, 101, 110, 111.
Note: if there is no don't care in the target, then behavior of Yes is same as No

Hybrid(Y/N):
Yes: performing both Genetic Algorithm guided and random logic sim
No: only performing search using random logic simulation

Number of node to expand: enter integer
Number of node N to expand before termination. For hybrid approach, logic simulation will expand N number of node, then GA will expand N/2 number of node. A typical number of node to expand depends on the size of the circuit. For apoptosis, it is roughly 300 nodes or more.
If it is a hybrid approach, the program will ask for mutation rate, specify in integer. A typical mutation rate 6 => 6%

Rate Search(Y/N):
Yes: performing a rate substitution. Note, only choose Yes if in the init file row 3, there are PI that is specified as Xs, and row 5 is not zero.
No: do regular search
If Rate Search is selected as Yes, the following question will be asked:
Input index: enter integer <= asking which rate to enumerate.
For example if row 5 value is 3. Then if user want to enumerate the first rate on row 6, its index is 0, second rate on row 7, its index is 1 and so on
Lower range: starting value of the rate to enumerate from
Upper range: final value of the rate to enumerate to
Increment step: step size of the enumeration
Then a regular search is performed for each rate value (take a very long time)

Output:
Apoptosis.log will contain all the result for the search.
Apoptosis.data contains quick summary of the result for each target set with:
Number of states reached
Number in closed list
Number in reached list
Percent reached from set of target

NOTE: *to perform extended search, where each sequence that found a new state is simulated for an extended period of cycles until the system reach a steady state. During compile time add –D EXTEND option*

A.2.2 SAT-based Verification Methods

This method will require:
Apoptosis.lev: circuit in netlist format
Apoptosis.init: initialization file
Apoptosis.target: target state to be searched

The format and layout is exactly the same as for Reachablity Analysis above.
This requires that zChaff is installed, and the executable path is added to environment variable path. Similarly, a cnf preprocessing software hypre also need to be installed and its executable path is also added to the environment.

NOTE MUST READ: when compile, -D RESET option should be turn on. Allow the assumption that the last pin is an active low reset pin. A constraint clause is added to keep this signal at 1 for all time. Hence this compiled executable can not be used for other circuit unless recompiled is done without –D RESET

The flowing are different command line argument pattern

./a.exe <.lev> <Time-frame> <Time-out>
Unroll the circuit for the number of Time-frame and inject initial condition and target constraints to perform BMC on each target listed in .target file until key word "END"

./a.exe <lev> <Time-frame> 0
Unroll the circuit for the number of Time-frame without any properties injection

*./a.exe <.lev> <Time-frame> <Time-out> -1**
Unroll, for each row in target file, enumerate through all the specified bit. For example if target is 0X0, this command will perform 4 BMCs on these following targets: 0X0, 0X1, 1X0, 1X1;
*This option also accepts another file as input with extension .reached. For example: .
./a.exe Apoptosis 5 100 -1 will enumerate through each target. If Apoptosis.reached is available this can speed up the process
Apoptosis.reach is sets of a list of reachable state. If in the Apoptosis.target file have 6 targets in it set, then Apoptosis.reach need to have 6 set of list of reachable state. Each set starts off with the number of state in that list and ends with the keyword "END". Each state is listed on a separate line. Apoptosis.reach is generated by GALS tool. If this Apoptosis.reached file is available, when the program enumerate through each value of the target, it will check against this list, if it is in the list, the program will skip this value of target and move on to the next enumeration.

./a.exe <.lev> <Time-frame> <Time-out> -1 Apoptosis_T5.cnf.enum
Same command as the above description, with an addition of a hint file
After executing the command without hint file for 5 time frames.. This will generate an output file Apoptosis_T5.cnf.enum. Now says you wish to run the same command but now with 7 time frames. For those target that is UNSAT in 5 time frames will also be UNSAT in 7. By providing the file name that contains the output from previous run of 5 time frames, this run with 7 time frames will skip over target that was UNSAT in 5 time frames.

./a.exe <.lev> <Time-frame> <Time-out> -2
Unroll, read in .properties file (from Aborted states ITC 99), does not need .init or .target
Perform BMC

./a.exe <.lev> <Time-frame> <Time-out> -3
Unroll, read in .properties file (from Aborted states ITC 99), does not need .init or .target
Perform Induction

./a.exe <.lev> <Time-frame (total) > <Time-out> <Time-Frame known>
This is to perform incremental verification. TF total is how many time frames to unroll. TF known need to be < TF total, it specified that from 0 to TF known, target is not reachable so p is injected in TF 0 to TF known, and p is injected from TF known +1 to TF total.

Output files:
Example Command
./BMC.exe Apoptosis 5 100 -1
Will reading Apoptosis.lev, Apoptosis.init, Apoptosis.target and create:
Apoptosis_T5.cnf: cnf file of the circuit unrolling 5 time frames with properties injected
Apoptosis_T5.cnf.enum: result of the enumeration of target
Apoptosis_T5.cnf.satOut: output of zChaff
If Apoptosis.reached is available, it will speed up the process

A.3 Fault Models Construction Instructions

Fault models construction & mapping
Single and Double Faults
.vec has the following format:
numInput, resolution
init state
input Sequence
. . . .
END **Single stuck at fault injection simulation**

Need: .lev, .vec
Create : Apoptosis_single.simu
./FaultSim.exe Apoptosis
This execution reads in Apoptosis.lev file, goes through every gate, ignoring INPUT, OUTPUT, BUFFER and NOT gates, and simulate both s-a 0 and s-a 1 for the gate using the input sequence stored in Apoptosis.vec, the final flip flop concentration is printed to file.
NOTE: when compile, if –D SIM option is turn on, each time step of the simulation is printed to file, instead of just then end, along with some internal gate values specified in the constructer.
//rename this file, for example Apop122.sim **Grouping fault pattern**

Use grep to grab similar output response faults, and put in a .simFault file
For example
grep " 0) " Apop122.sim > Apop122Zero.simFault
Grab all simulation with tBID_cyto remain zero **Mapping for single stuck at fault**

1. **Create Mapping file** (Only need to be done once)

   Need Apoptosis.pin, Apoptosis.name file

   Create: Apoptosis.map

.pin listing of all mapping between blif pin and wire name, .pin need to be manually generated by looking at correlation between wire name and pin number in Apoptosis_full.blif and Apoptosis.blif

.name generate when convert from benched to lev

./Mapping Apoptosis 1 trash

This command will merge Apoptosis.name and Apoptosis.pin to create Apoptosis.map for the circuit

2. **Generate set of equivalent fault**

   Need: .simFault, .eqf

   Create: .toMap

   Given a list of fault in a file with extension of .simFault

   ./Mapping Apoptosis 2 Apop122Zero

   This command will generate a file Apop122Zero.map, which contains a list of equivalent set of fault. The program traverse each fault in the list, find all of it equivalent fault. Eliminate any fault in the list if it is already in the equivalent set. Also ignore any redundant fault (RED). Equivalent set if written to .toMap file.

3. **Mapping the set of equivalent fault**

   Need: .toMap, .map (generate in step 1)

   Create: .mapped

   ./Mapping Apoptosis 3 Apop122Zero > Apop122Zero.map

   Go through each fault in the equivalent set, attempt to map the gate back to original wire name by accessing .map file. Result is written to Apop122Zero.map. For each equivalent set that has a least a gate mapped, the last gate number and wire name is written to Apop122Zero.mapped.

Double fault model construction
Need: .vec, .mapped (from step 3)
Create: .simu
./FaultSim Apoptosis Apop122Norm
This command goes through each fault in the Apop122Norm.mapped file, pair that fault up with the rest of the decending faults and perform double s-a fault simulation. The flip flop end result is written in Apop122Norm.simu.
Rename Apop122Norm.simu to Apop2F.sim **Grouping fault pattern similar to above**

grep " 0)" Apop2F.sim > Apop2FZero.simFault **Mapping double s-a fault**

Need: .simFault, .map
Create: cout
./Mapping Apoptosis 5 Apop2FZero
This command goes through each pair of fault in the Apop2FZero.simFault file, access Apoptosis.map file and map the corresponding wire name. The result is print to screen

Appendix B

Experimental Data

B.1 Reachability Analysis

B.1.1 Stand Alone BID Model

tBID_cyto: right most bits
Two representative sequence for tBID_cyto concentration that reach value 19 and 23

Input sequence:

XXXXXXXXXXXXXXXX (0 0)	XXXXXXXXXXXXXXXXXXXXXXXX (0 0 0)
1101001101111000 (211 120)	000000111111101000000000 (3 250 0)
1101001101111000 (211 120)	001111101011111100000000 (62 191 0)
1101001101111000 (211 120)	011110001000001100000010 (120 131 2)
0000000111010000 (1 208)	101010010100111100000101 (169 79 5)
1101001101111000 (211 120)	110111000001100000001001 (220 24 9)
1101001101111000 (211 120)	111000000000111100001110 (224 15 14)
0000000111010000 (1 208)	111000110000100100010001 (227 9 17)
	111001110000001100010011 (231 3 19)

Input sequence:

XXXXXXXXXXXXXXXX (0 0)	XXXXXXXXXXXXXXXXXXXXXXXX (0 0 0)
1101001101111000 (211 120)	000000111111101000000000 (3 250 0)
1101001101111000 (211 120)	001111101011111100000000 (62 191 0)
0000000111010000 (1 208)	011110001000001100000010 (120 131 2)
0011010001101011 (52 107)	110011110010100100000101 (207 41 5)
0110101001111110 (106 126)	110110000001101100001010 (216 27 10)
1101001101111000 (211 120)	110111100001000000001111 (222 16 15)
1101001101111000 (211 120)	111000100000100100010010 (226 9 18)
0011010001101011 (52 107)	111000110000011000010100 (227 6 20)

1101001101111000 (211 120)
0100100011101100 (72 236)
1110010100000011000101010 (229 3 21)
1110010100000010000101110 (229 2 22)
1110011000000000000010111 (230 0 23)

B.1.2 Complete Model

tBID cytoplasm input sequence

Input sequence:
00000010 (2) 0111110100000000111110100000000000000000000000011111110100000000 (125 0 250 0 0 3 250 0)
10010100 (148) 0111110000000000111110000000001000000000000000010111110110000000 (124 0 248 2 0 2 251 0)
11000000 (192) 0111101100000000011111100111110000000000000000001111111000000000 (123 0 126 124 0 1 252 0)
01111011 (123) 0111101100000100001011011100100100000000000000001111111000000000 (123 4 45 201 0 1 252 0)
10101110 (174) 0111100100000111000110101101010100000100000000111111101000000000 (121 7 26 213 4 3 250 0)
01111011 (123) 0111010100001000000010111101110100001010000001011111100000000000 (117 8 11 221 10 5 248 0)
00011001 (25) 0111000000001001000001111101100100010001000001111111011000000000 (112 9 7 217 17 7 246 0)
00011001 (25) 0110101000001010000001101101001000011000000010001111010100000000 (106 10 6 210 24 8 245 0)
01110100 (116) 0110010000001001000001011100110000100000000010101111001100000000 (100 9 5 204 32 10 243 0)
00101000 (40) 0101111000001001000000111100011100100111000010111111001000000000 (94 9 3 199 39 11 242 0)
01101000 (104) 0101100000001001000000111100000000101110000011001111000100000000 (88 9 3 192 46 12 241 0)
11011101 (221) 0101001000001001000000101011101000110101000011011111000000000000 (82 9 2 186 53 13 240 0)
11100100 (228) 0100110000001001000000001011010100111100000011101110111100000000 (76 9 0 181 60 14 239 0)
11100100 (228) 0100011000001000000000001010111101000011000011111110111000000000 (70 8 0 175 67 15 238 0)
11101011 (235) 0100000100001000000000001010100101001001000011111110110100000001 (65 8 0 169 73 15 237 1)
00101000 (40) 0011110000001000000000001010001101001111000011111110110000000010 (60 8 0 163 79 15 236 2)

01110100 (116) 00110111000010000000000010011101010101010001000011101011000000010 (55 8 0 157 85 16 235 2)
11011010 (218) 00110010000010000000000010010111010110110001000111101010000000010 (50 8 0 151 91 17 234 2)
01101000 (104) 00101101000001110000000010010010011000010001001011101001000000010 (45 7 0 146 97 18 233 2)
11110011 (243) 00101001000010000000000010001101011001010001001011101001000000010 (41 8 0 141 101 18 233 2)
00011001 (25) 00100101000010000000000010001000011010100001001111101000000000010 (37 8 0 136 106 19 232 2)
01110100 (116) 00100001000010000000000010000011011011110001010011100111000000010 (33 8 0 131 111 20 231 2)
11110011 (243) 00011101000010000000000001111110011101000001010111100110000000010 (29 8 0 126 116 21 230 2)
00011001 (25) 00011010000001110000000001111010011110010001011111100100000000010 (26 7 0 122 121 23 228 2)
00011001 (25) 00010111000001110000000001110110011111010001100011100011000000010 (23 7 0 118 125 24 227 2)
11110011 (243) 00010100000001110000000001110010100000010001100111100010000000010 (20 7 0 114 129 25 226 2)
00011001 (25) 00010001000001110000000001101110100001010001101011100001000000010 (17 7 0 110 133 26 225 2)
01110100 (116) 00001111000010000000000001101010100010000001101111100000000000010 (15 8 0 106 136 27 224 2)
11011101 (221) 00001101000010000000000001100110100011000001110111011110000000010 (13 8 0 102 140 29 222 2)
01110100 (116) 00001100000010000000000001100010100100000010000011011011000000010 (12 8 0 98 144 32 219 2)
11110011 (243) 00001010000001110000000001011111100101000010001011011001000000010 (10 7 0 95 148 34 217 2)
00011001 (25) 00001001000001110000000001011100100101110010010011010111000000010 (9 7 0 92 151 36 215 2)
11110011 (243) 00001000000001110000000001011001100110100010011011010100000010 (8 7 0 89 154 38 213 2)
00011001 (25) 00000111000001110000000001010110100111010010100011010011000000010 (7 7 0 86 157 40 211 2)
00000000 (0) 00000110000001110000000001010011101000000010101011010001000000010 (6 7 0 83 160 42 209 2)
00000000 (0) 00000101000001110000000001010000101000110010110011001111000000010 (5 7 0 80 163 44 207 2)
00000000 (0) 00000100000001110000000001001101101001100010110111001101000000011 (4 7 0 77 166 45 205 3)

00000000 (0) 00000011000001110000000001001010101010010010111011001011000001004 (3 7 0 74 169 46 203 4)
00000000 (0) 00000010000001110000000001000111101011000010111111001001000001015 (2 7 0 71 172 47 201 5)
00000000 (0) 00000010000001100000000001000101101011110011001011000110000001015 (2 6 0 69 175 50 198 5)
00000000 (0) 00000010000001010000000001000011101100100011010111000011000001015 (2 5 0 67 178 53 195 5)
00000000 (0) 00000010000001010000000001000001101101000011011111000001000001015 (2 5 0 65 180 55 193 5)
00000000 (0) 00000010000001010000000000111111101101100011100110111111000001015 (2 5 0 63 182 57 191 5)
00000000 (0) 00000010000001010000000000111101101110000011101110111101000001015 (2 5 0 61 184 59 189 5)
00000000 (0) 00000010000001010000000000111011101110100011110110111011000001015 (2 5 0 59 186 61 187 5)
00000000 (0) 00000010000001010000000000111001101111000011111110111001000001015 (2 5 0 57 188 63 185 5)
00000000 (0) 00000010000001010000000000110111101111100100000110110111000001015 (2 5 0 55 190 65 183 5)
00000000 (0) 00000001000001010000000000110101110000000100001010110110000001015 (1 5 0 53 192 66 182 5)
00000000 (0) 00000001000001010000000000110011110000100100010010110100000001015 (1 5 0 51 194 68 180 5)
00000000 (0) 00000001000001010000000000110001110001000100011010110010000001015 (1 5 0 49 196 70 178 5)
00000000 (0) 00000001000001010000000000101111110001100100100010110000000001015 (1 5 0 47 198 72 176 5)
00000000 (0) 00000001000001010000000000101101110010000100100110101110000001106 (1 5 0 45 200 73 174 6)
00000000 (0) 00000001000001010000000000101011110010100100101010101100000001117 (1 5 0 43 202 74 172 7)
00000000 (0) 00000001000001010000000000101001110011000100101110101010000010008 (1 5 0 41 204 75 170 8)
00000000 (0) 00000001000001000000000000101000110011100100110110101000000010008 (1 4 0 40 206 77 168 8)
00000000 (0) 00000001000000110000000000100111110100000100111110100110000010008 (1 3 0 39 208 79 166 8)
00000000 (0) 00000001000000110000000000100110110100010101000010100101000010008 (1 3 0 38 209 80 165 8)
00000000 (0) 00000001000000110000000000100101110100100101000110100100000010008 (1 3 0 37 210 81 164 8)

00000000 (0) 0000000100000011000000000001001001101001101010010101000110000100 (1 3 0 36 211 82 163 8)
00000000 (0) 0000000100000011000000000001000111101010001010011101000100000100 (1 3 0 35 212 83 162 8)
00000000 (0) 0000000100000011000000000001000101101010101010100101000010000100 (1 3 0 34 213 84 161 8)
00000000 (0) 0000000100000011000000000001000011101011001010101101000000000100 (1 3 0 33 214 85 160 8)
00000000 (0) 0000000100000011000000000001000001101011101010110100111110000100 (1 3 0 32 215 86 159 8)
00000000 (0) 0000000100000011000000000000111111101100001010111001111000000100 (1 3 0 31 216 87 158 8)
00000000 (0) 0000000100000011000000000000111101101100101011000100111010000100 (1 3 0 30 217 88 157 8)
00000000 (0) 0000000100000011000000000000111011101101001011001100111000000100 (1 3 0 29 218 89 156 8)
00000000 (0) 0000000100000011000000000000111001101101101011010100110110000100 (1 3 0 28 219 90 155 8)
00000000 (0) 0000000100000011000000000000110111101110001011011100110100000100 (1 3 0 27 220 91 154 8)
00000000 (0) 0000000100000011000000000000110101101110101011100100110010000100 (1 3 0 26 221 92 153 8)
00000000 (0) 0000000100000011000000000000110011101111001011101100110000000100 (1 3 0 25 222 93 152 8)
00000000 (0) 0000000100000011000000000000110001101111101011110100101110000100 (1 3 0 24 223 94 151 8)
00000000 (0) 0000000100000011000000000000101111110000001011111100101100000100 (1 3 0 23 224 95 150 8)
00000000 (0) 0000000100000011000000000000101101110000101100000100101010000100 (1 3 0 22 225 96 149 8)
00000000 (0) 0000000100000011000000000000101011110001001100001100101000000100 (1 3 0 21 226 97 148 8)
00000000 (0) 0000000100000011000000000000101001110001101100010100100110000100 (1 3 0 20 227 98 147 8)
00000000 (0) 0000000100000011000000000000100111110010001100011100100100000100 (1 3 0 19 228 99 146 8)
00000000 (0) 0000000100000011000000000000100101110010101100100100100010000100 (1 3 0 18 229 100 145 8)
00000000 (0) 0000000100000011000000000000100011110011001100100100100000000101 (1 3 0 17 230 100 144 9)
00000000 (0) 0000000100000011000000000000100001110011101100100100011110000100 (1 3 0 16 231 100 143 10)

00000000 (0) 00000001000000110000000000000111111101000011001001000111000001011 (1 3 0 15 232 100 142 11)
00000000 (0) 00000001000000110000000000000111011101001011001011000110100001011 (1 3 0 14 233 101 141 11)
00000000 (0) 00000001000000100000000000000111011101010011001101000110000001011 (1 2 0 14 234 102 140 11)
00000001000000010000000000000111011101011011001111000101100001011 (1 1 0 14 235 103 139 11)

B.2 SAT-based Verification

B.2.1 Stand Alone BID Model

tBID in cytoplasm induction case result
All base cases are UNSAT except for the reachable ones

00000000	Reachable(OLD)
00000001	Reachable(OLD)
00000010	Reachable(OLD)
00000011	Reachable(OLD)
00000100	Reachable(OLD)
00000101	Reachable(OLD)
00000110	Reachable(OLD)
00000111	Reachable(OLD)
00001000	Reachable(OLD)
00001001	Reachable(OLD)
00001010	Reachable(OLD)
00001011	Reachable(OLD)
00001100	Reachable(OLD)
00001101	Reachable(OLD)
00001110	Reachable(OLD)
00001111	Reachable(OLD)
00010000	Reachable(OLD)
00010001	Reachable(OLD)
00010010	Reachable(OLD)
00010011	Reachable(OLD)
00010100	Reachable(OLD)
00010101	Reachable(OLD)
00010110	Reachable(OLD)
00010111	Reachable(OLD)
00011000	RESULT: SAT
00011001	RESULT: SAT
00011010	RESULT: SAT
00011011	RESULT: UNSAT
00011100	RESULT: UNSAT
00011101	RESULT: UNSAT
00011110	RESULT: UNSAT
00011111	RESULT: UNSAT
00100000	RESULT: UNSAT(OLD)
00100001	RESULT: UNSAT(OLD)
00100010	RESULT: UNSAT(OLD)
00100011	RESULT: UNSAT(OLD)
00100100	RESULT: UNSAT(OLD)
00100101	RESULT: UNSAT(OLD)
00100110	RESULT: UNSAT(OLD)
00100111	RESULT: UNSAT(OLD)
00101000	RESULT: UNSAT(OLD)
00101001	RESULT: UNSAT(OLD)
00101010	RESULT: UNSAT(OLD)
00101011	RESULT: UNSAT(OLD)
00101100	RESULT: UNSAT(OLD)
00101101	RESULT: UNSAT(OLD)
00101110	RESULT: UNSAT(OLD)
00101111	RESULT: UNSAT(OLD)
00110000	RESULT: UNSAT(OLD)
00110001	RESULT: UNSAT(OLD)
00110010	RESULT: UNSAT(OLD)

00110011 RESULT: UNSAT(OLD)
00110100 RESULT: UNSAT(OLD)
00110101 RESULT: UNSAT(OLD)
00110110 RESULT: UNSAT(OLD)
00110111 RESULT: UNSAT(OLD)
00111000 RESULT: UNSAT(OLD)
00111001 RESULT: UNSAT(OLD)
00111010 RESULT: UNSAT(OLD)
00111011 RESULT: UNSAT(OLD)
00111100 RESULT: UNSAT(OLD)
00111101 RESULT: UNSAT(OLD)
00111110 RESULT: UNSAT(OLD)
00111111 RESULT: UNSAT(OLD)
01000000 RESULT: UNSAT(OLD)
01000001 RESULT: UNSAT(OLD)
01000010 RESULT: UNSAT(OLD)
01000011 RESULT: UNSAT(OLD)
01000100 RESULT: UNSAT(OLD)
01000101 RESULT: UNSAT(OLD)
01000110 RESULT: UNSAT(OLD)
01000111 RESULT: UNSAT(OLD)
01001000 RESULT: UNSAT(OLD)
01001001 RESULT: UNSAT(OLD)
01001010 RESULT: UNSAT(OLD)
01001011 RESULT: UNSAT(OLD)
01001100 RESULT: UNSAT(OLD)
01001101 RESULT: UNSAT(OLD)
01001110 RESULT: UNSAT(OLD)
01001111 RESULT: UNSAT(OLD)
01010000 RESULT: UNSAT(OLD)
01010001 RESULT: UNSAT(OLD)
01010010 RESULT: UNSAT(OLD)
01010011 RESULT: UNSAT(OLD)
01010100 RESULT: UNSAT(OLD)
01010101 RESULT: UNSAT(OLD)
01010110 RESULT: UNSAT(OLD)
01010111 RESULT: UNSAT(OLD)
01011000 RESULT: UNSAT(OLD)
01011001 RESULT: UNSAT(OLD)
01011010 RESULT: UNSAT(OLD)
01011011 RESULT: UNSAT(OLD)
01011100 RESULT: UNSAT(OLD)
01011101 RESULT: UNSAT(OLD)
01011110 RESULT: UNSAT(OLD)
01011111 RESULT: UNSAT(OLD)
01100000 RESULT: UNSAT(OLD)
01100001 RESULT: UNSAT(OLD)
01100010 RESULT: UNSAT(OLD)
01100011 RESULT: UNSAT(OLD)
01100100 RESULT: UNSAT(OLD)
01100101 RESULT: UNSAT(OLD)
01100110 RESULT: UNSAT(OLD)
01100111 RESULT: UNSAT(OLD)
01101000 RESULT: UNSAT(OLD)
01101001 RESULT: UNSAT(OLD)
01101010 RESULT: UNSAT(OLD)
01101011 RESULT: UNSAT(OLD)
01101100 RESULT: UNSAT(OLD)
01101101 RESULT: UNSAT(OLD)
01101110 RESULT: UNSAT(OLD)
01101111 RESULT: UNSAT(OLD)
01110000 RESULT: UNSAT(OLD)
01110001 RESULT: UNSAT(OLD)
01110010 RESULT: UNSAT(OLD)
01110011 RESULT: UNSAT(OLD)
01110100 RESULT: UNSAT(OLD)
01110101 RESULT: UNSAT(OLD)
01110110 RESULT: UNSAT(OLD)
01110111 RESULT: UNSAT(OLD)
01111000 RESULT: UNSAT(OLD)
01111001 RESULT: UNSAT(OLD)
01111010 RESULT: UNSAT(OLD)
01111011 RESULT: UNSAT(OLD)
01111100 RESULT: UNSAT(OLD)
01111101 RESULT: UNSAT(OLD)
01111110 RESULT: UNSAT(OLD)
01111111 RESULT: UNSAT(OLD)
10000000 RESULT: UNSAT(OLD)
10000001 RESULT: UNSAT(OLD)
10000010 RESULT: UNSAT(OLD)
10000011 RESULT: UNSAT(OLD)
10000100 RESULT: UNSAT(OLD)
10000101 RESULT: UNSAT(OLD)
10000110 RESULT: UNSAT(OLD)

10000111 RESULT: UNSAT(OLD)
10001000 RESULT: UNSAT(OLD)
10001001 RESULT: UNSAT(OLD)
10001010 RESULT: UNSAT(OLD)
10001011 RESULT: UNSAT(OLD)
10001100 RESULT: UNSAT(OLD)
10001101 RESULT: UNSAT(OLD)
10001110 RESULT: UNSAT(OLD)
10001111 RESULT: UNSAT(OLD)
10010000 RESULT: UNSAT(OLD)
10010001 RESULT: UNSAT(OLD)
10010010 RESULT: UNSAT(OLD)
10010011 RESULT: UNSAT(OLD)
10010100 RESULT: UNSAT(OLD)
10010101 RESULT: UNSAT(OLD)
10010110 RESULT: UNSAT(OLD)
10010111 RESULT: UNSAT(OLD)
10011000 RESULT: UNSAT(OLD)
10011001 RESULT: UNSAT(OLD)
10011010 RESULT: UNSAT(OLD)
10011011 RESULT: UNSAT(OLD)
10011100 RESULT: UNSAT(OLD)
10011101 RESULT: UNSAT(OLD)
10011110 RESULT: UNSAT(OLD)
10011111 RESULT: UNSAT(OLD)
10100000 RESULT: UNSAT(OLD)
10100001 RESULT: UNSAT(OLD)
10100010 RESULT: UNSAT(OLD)
10100011 RESULT: UNSAT(OLD)
10100100 RESULT: UNSAT(OLD)
10100101 RESULT: UNSAT(OLD)
10100110 RESULT: UNSAT(OLD)
10100111 RESULT: UNSAT(OLD)
10101000 RESULT: UNSAT(OLD)
10101001 RESULT: UNSAT(OLD)
10101010 RESULT: UNSAT(OLD)
10101011 RESULT: UNSAT(OLD)
10101100 RESULT: UNSAT(OLD)
10101101 RESULT: UNSAT(OLD)
10101110 RESULT: UNSAT(OLD)
10101111 RESULT: UNSAT(OLD)
10110000 RESULT: UNSAT(OLD)
10110001 RESULT: UNSAT(OLD)
10110010 RESULT: UNSAT(OLD)
10110011 RESULT: UNSAT(OLD)
10110100 RESULT: UNSAT(OLD)
10110101 RESULT: UNSAT(OLD)
10110110 RESULT: UNSAT(OLD)
10110111 RESULT: UNSAT(OLD)
10111000 RESULT: UNSAT(OLD)
10111001 RESULT: UNSAT(OLD)
10111010 RESULT: UNSAT(OLD)
10111011 RESULT: UNSAT(OLD)
10111100 RESULT: UNSAT(OLD)
10111101 RESULT: UNSAT(OLD)
10111110 RESULT: UNSAT(OLD)
10111111 RESULT: UNSAT(OLD)
11000000 RESULT: UNSAT(OLD)
11000001 RESULT: UNSAT(OLD)
11000010 RESULT: UNSAT(OLD)
11000011 RESULT: UNSAT(OLD)
11000100 RESULT: UNSAT(OLD)
11000101 RESULT: UNSAT(OLD)
11000110 RESULT: UNSAT(OLD)
11000111 RESULT: UNSAT(OLD)
11001000 RESULT: UNSAT(OLD)
11001001 RESULT: UNSAT(OLD)
11001010 RESULT: UNSAT(OLD)
11001011 RESULT: UNSAT(OLD)
11001100 RESULT: UNSAT(OLD)
11001101 RESULT: UNSAT(OLD)
11001110 RESULT: UNSAT(OLD)
11001111 RESULT: UNSAT(OLD)
11010000 RESULT: UNSAT(OLD)
11010001 RESULT: UNSAT(OLD)
11010010 RESULT: UNSAT(OLD)
11010011 RESULT: UNSAT(OLD)
11010100 RESULT: UNSAT(OLD)
11010101 RESULT: UNSAT(OLD)
11010110 RESULT: UNSAT(OLD)
11010111 RESULT: UNSAT(OLD)
11011000 RESULT: UNSAT(OLD)
11011001 RESULT: UNSAT(OLD)
11011010 RESULT: UNSAT(OLD)

11011011	RESULT: UNSAT(OLD)
11011100	RESULT: UNSAT(OLD)
11011101	RESULT: UNSAT(OLD)
11011110	RESULT: UNSAT(OLD)
11011111	RESULT: UNSAT(OLD)
11100000	RESULT: UNSAT(OLD)
11100001	RESULT: UNSAT(OLD)
11100010	RESULT: UNSAT(OLD)
11100011	RESULT: UNSAT(OLD)
11100100	RESULT: UNSAT(OLD)
11100101	RESULT: UNSAT(OLD)
11100110	RESULT: UNSAT(OLD)
11100111	RESULT: UNSAT(OLD)
11101000	RESULT: UNSAT(OLD)
11101001	RESULT: UNSAT(OLD)
11101010	RESULT: UNSAT(OLD)
11101011	RESULT: UNSAT(OLD)
11101100	RESULT: UNSAT(OLD)
11101101	RESULT: UNSAT(OLD)
11101110	RESULT: UNSAT(OLD)
11101111	RESULT: UNSAT(OLD)
11110000	RESULT: UNSAT(OLD)
11110001	RESULT: UNSAT(OLD)
11110010	RESULT: UNSAT(OLD)
11110011	RESULT: UNSAT(OLD)
11110100	RESULT: UNSAT(OLD)
11110101	RESULT: UNSAT(OLD)
11110110	RESULT: UNSAT(OLD)
11110111	RESULT: UNSAT(OLD)
11111000	RESULT: UNSAT(OLD)
11111001	RESULT: UNSAT(OLD)
11111010	RESULT: UNSAT(OLD)
11111011	RESULT: UNSAT(OLD)
11111100	RESULT: UNSAT(OLD)
11111101	RESULT: UNSAT(OLD)
11111110	RESULT: UNSAT(OLD)
11111111	RESULT: UNSAT(OLD)

B.2.2 Complete Model

tBID in the cytoplasm induction case The base case is all UNSAT except for the reachable ones

00000000	Reachable(OLD)
00000001	Reachable(OLD)
00000010	Reachable(OLD)
00000011	Reachable(OLD)
00000100	Reachable(OLD)
00000101	Reachable(OLD)
00000110	Reachable(OLD)
00000111	Reachable(OLD)
00001000	Reachable(OLD)
00001001	Reachable(OLD)
00001010	Reachable(OLD)
00001011	Reachable(OLD)
00001100	RESULT: ABORT
00001101	RESULT: SAT
00001110	RESULT: ABORT
00001111	RESULT: ABORT
00010000	RESULT: ABORT
00010001	RESULT: ABORT
00010010	RESULT: ABORT
00010011	RESULT: ABORT
00010100	RESULT: ABORT
00010101	RESULT: ABORT
00010110	RESULT: ABORT
00010111	RESULT: ABORT
00011000	RESULT: ABORT
00011001	RESULT: ABORT
00011010	RESULT: ABORT
00011011	RESULT: ABORT
00011100	RESULT: SAT
00011101	RESULT: SAT
00011110	RESULT: SAT
00011111	RESULT: SAT
00100000	RESULT: UNSAT
00100001	RESULT: UNSAT

00100010 RESULT: UNSAT
00100011 RESULT: UNSAT
00100100 RESULT: UNSAT
00100101 RESULT: UNSAT
00100110 RESULT: UNSAT
00100111 RESULT: UNSAT
00101000 RESULT: UNSAT
00101001 RESULT: UNSAT(OLD)
00101010 RESULT: UNSAT(OLD)
00101011 RESULT: UNSAT(OLD)
00101100 RESULT: UNSAT(OLD)
00101101 RESULT: UNSAT(OLD)
00101110 RESULT: UNSAT(OLD)
00101111 RESULT: UNSAT(OLD)
00110000 RESULT: UNSAT(OLD)
00110001 RESULT: UNSAT(OLD)
00110010 RESULT: UNSAT(OLD)
00110011 RESULT: UNSAT(OLD)
00110100 RESULT: UNSAT(OLD)
00110101 RESULT: UNSAT(OLD)
00110110 RESULT: UNSAT(OLD)
00110111 RESULT: UNSAT(OLD)
00111000 RESULT: UNSAT(OLD)
00111001 RESULT: UNSAT(OLD)
00111010 RESULT: UNSAT(OLD)
00111011 RESULT: UNSAT(OLD)
00111100 RESULT: UNSAT(OLD)
00111101 RESULT: UNSAT(OLD)
00111110 RESULT: UNSAT(OLD)
00111111 RESULT: UNSAT(OLD)
01000000 RESULT: UNSAT(OLD)
01000001 RESULT: UNSAT(OLD)
01000010 RESULT: UNSAT(OLD)
01000011 RESULT: UNSAT(OLD)
01000100 RESULT: UNSAT(OLD)
01000101 RESULT: UNSAT(OLD)
01000110 RESULT: UNSAT(OLD)
01000111 RESULT: UNSAT(OLD)
01001000 RESULT: UNSAT(OLD)
01001001 RESULT: UNSAT(OLD)
01001010 RESULT: UNSAT(OLD)
01001011 RESULT: UNSAT(OLD)
01001100 RESULT: UNSAT(OLD)
01001101 RESULT: UNSAT(OLD)
01001110 RESULT: UNSAT(OLD)
01001111 RESULT: UNSAT(OLD)
01010000 RESULT: UNSAT(OLD)
01010001 RESULT: UNSAT(OLD)
01010010 RESULT: UNSAT(OLD)
01010011 RESULT: UNSAT(OLD)
01010100 RESULT: UNSAT(OLD)
01010101 RESULT: UNSAT(OLD)
01010110 RESULT: UNSAT(OLD)
01010111 RESULT: UNSAT(OLD)
01011000 RESULT: UNSAT(OLD)
01011001 RESULT: UNSAT(OLD)
01011010 RESULT: UNSAT(OLD)
01011011 RESULT: UNSAT(OLD)
01011100 RESULT: UNSAT(OLD)
01011101 RESULT: UNSAT(OLD)
01011110 RESULT: UNSAT(OLD)
01011111 RESULT: UNSAT(OLD)
01100000 RESULT: UNSAT(OLD)
01100001 RESULT: UNSAT(OLD)
01100010 RESULT: UNSAT(OLD)
01100011 RESULT: UNSAT(OLD)
01100100 RESULT: UNSAT(OLD)
01100101 RESULT: UNSAT(OLD)
01100110 RESULT: UNSAT(OLD)
01100111 RESULT: UNSAT(OLD)
01101000 RESULT: UNSAT(OLD)
01101001 RESULT: UNSAT(OLD)
01101010 RESULT: UNSAT(OLD)
01101011 RESULT: UNSAT(OLD)
01101100 RESULT: UNSAT(OLD)
01101101 RESULT: UNSAT(OLD)
01101110 RESULT: UNSAT(OLD)
01101111 RESULT: UNSAT(OLD)
01110000 RESULT: UNSAT(OLD)
01110001 RESULT: UNSAT(OLD)
01110010 RESULT: UNSAT(OLD)
01110011 RESULT: UNSAT(OLD)
01110100 RESULT: UNSAT(OLD)
01110101 RESULT: UNSAT(OLD)

01110110 RESULT: UNSAT(OLD)
01110111 RESULT: UNSAT(OLD)
01111000 RESULT: UNSAT(OLD)
01111001 RESULT: UNSAT(OLD)
01111010 RESULT: UNSAT(OLD)
01111011 RESULT: UNSAT(OLD)
01111100 RESULT: UNSAT(OLD)
01111101 RESULT: UNSAT(OLD)
01111110 RESULT: UNSAT(OLD)
01111111 RESULT: UNSAT(OLD)
10000000 RESULT: UNSAT(OLD)
10000001 RESULT: UNSAT(OLD)
10000010 RESULT: UNSAT(OLD)
10000011 RESULT: UNSAT(OLD)
10000100 RESULT: UNSAT(OLD)
10000101 RESULT: UNSAT(OLD)
10000110 RESULT: UNSAT(OLD)
10000111 RESULT: UNSAT(OLD)
10001000 RESULT: UNSAT(OLD)
10001001 RESULT: UNSAT(OLD)
10001010 RESULT: UNSAT(OLD)
10001011 RESULT: UNSAT(OLD)
10001100 RESULT: UNSAT(OLD)
10001101 RESULT: UNSAT(OLD)
10001110 RESULT: UNSAT(OLD)
10001111 RESULT: UNSAT(OLD)
10010000 RESULT: UNSAT(OLD)
10010001 RESULT: UNSAT(OLD)
10010010 RESULT: UNSAT(OLD)
10010011 RESULT: UNSAT(OLD)
10010100 RESULT: UNSAT(OLD)
10010101 RESULT: UNSAT(OLD)
10010110 RESULT: UNSAT(OLD)
10010111 RESULT: UNSAT(OLD)
10011000 RESULT: UNSAT(OLD)
10011001 RESULT: UNSAT(OLD)
10011010 RESULT: UNSAT(OLD)
10011011 RESULT: UNSAT(OLD)
10011100 RESULT: UNSAT(OLD)
10011101 RESULT: UNSAT(OLD)
10011110 RESULT: UNSAT(OLD)
10011111 RESULT: UNSAT(OLD)
10100000 RESULT: UNSAT(OLD)
10100001 RESULT: UNSAT(OLD)
10100010 RESULT: UNSAT(OLD)
10100011 RESULT: UNSAT(OLD)
10100100 RESULT: UNSAT(OLD)
10100101 RESULT: UNSAT(OLD)
10100110 RESULT: UNSAT(OLD)
10100111 RESULT: UNSAT(OLD)
10101000 RESULT: UNSAT(OLD)
10101001 RESULT: UNSAT(OLD)
10101010 RESULT: UNSAT(OLD)
10101011 RESULT: UNSAT(OLD)
10101100 RESULT: UNSAT(OLD)
10101101 RESULT: UNSAT(OLD)
10101110 RESULT: UNSAT(OLD)
10101111 RESULT: UNSAT(OLD)
10110000 RESULT: UNSAT(OLD)
10110001 RESULT: UNSAT(OLD)
10110010 RESULT: UNSAT(OLD)
10110011 RESULT: UNSAT(OLD)
10110100 RESULT: UNSAT(OLD)
10110101 RESULT: UNSAT(OLD)
10110110 RESULT: UNSAT(OLD)
10110111 RESULT: UNSAT(OLD)
10111000 RESULT: UNSAT(OLD)
10111001 RESULT: UNSAT(OLD)
10111010 RESULT: UNSAT(OLD)
10111011 RESULT: UNSAT(OLD)
10111100 RESULT: UNSAT(OLD)
10111101 RESULT: UNSAT(OLD)
10111110 RESULT: UNSAT(OLD)
10111111 RESULT: UNSAT(OLD)
11000000 RESULT: UNSAT(OLD)
11000001 RESULT: UNSAT(OLD)
11000010 RESULT: UNSAT(OLD)
11000011 RESULT: UNSAT(OLD)
11000100 RESULT: UNSAT(OLD)
11000101 RESULT: UNSAT(OLD)
11000110 RESULT: UNSAT(OLD)
11000111 RESULT: UNSAT(OLD)
11001000 RESULT: UNSAT(OLD)
11001001 RESULT: UNSAT(OLD)

11001010 RESULT: UNSAT(OLD)
11001011 RESULT: UNSAT(OLD)
11001100 RESULT: UNSAT(OLD)
11001101 RESULT: UNSAT(OLD)
11001110 RESULT: UNSAT(OLD)
11001111 RESULT: UNSAT(OLD)
11010000 RESULT: UNSAT(OLD)
11010001 RESULT: UNSAT(OLD)
11010010 RESULT: UNSAT(OLD)
11010011 RESULT: UNSAT(OLD)
11010100 RESULT: UNSAT(OLD)
11010101 RESULT: UNSAT(OLD)
11010110 RESULT: UNSAT(OLD)
11010111 RESULT: UNSAT(OLD)
11011000 RESULT: UNSAT(OLD)
11011001 RESULT: UNSAT(OLD)
11011010 RESULT: UNSAT(OLD)
11011011 RESULT: UNSAT(OLD)
11011100 RESULT: UNSAT(OLD)
11011101 RESULT: UNSAT(OLD)
11011110 RESULT: UNSAT(OLD)
11011111 RESULT: UNSAT(OLD)
11100000 RESULT: UNSAT(OLD)
11100001 RESULT: UNSAT(OLD)
11100010 RESULT: UNSAT(OLD)
11100011 RESULT: UNSAT(OLD)
11100100 RESULT: UNSAT(OLD)
11100101 RESULT: UNSAT(OLD)
11100110 RESULT: UNSAT(OLD)
11100111 RESULT: UNSAT(OLD)
11101000 RESULT: UNSAT(OLD)
11101001 RESULT: UNSAT(OLD)
11101010 RESULT: UNSAT(OLD)
11101011 RESULT: UNSAT(OLD)
11101100 RESULT: UNSAT(OLD)
11101101 RESULT: UNSAT(OLD)
11101110 RESULT: UNSAT(OLD)
11101111 RESULT: UNSAT(OLD)
11110000 RESULT: UNSAT(OLD)
11110001 RESULT: UNSAT(OLD)
11110010 RESULT: UNSAT(OLD)
11110011 RESULT: UNSAT(OLD)
11110100 RESULT: UNSAT(OLD)
11110101 RESULT: UNSAT(OLD)
11110110 RESULT: UNSAT(OLD)
11110111 RESULT: UNSAT(OLD)
11111000 RESULT: UNSAT(OLD)
11111001 RESULT: UNSAT(OLD)
11111010 RESULT: UNSAT(OLD)
11111011 RESULT: UNSAT(OLD)
11111100 RESULT: UNSAT(OLD)
11111101 RESULT: UNSAT(OLD)
11111110 RESULT: UNSAT(OLD)
11111111 RESULT: UNSAT(OLD)

B.3 Fault Models Analysis

B.3.1 Single Stuck at Faults For TA

Single stuck at fault for sequence TA
these faults prevent apoptosis

13237 \BCL2/BCL2_reg[7]_in Li00 13237 Li00
13239 \BCL2/BCL2_reg[5]_in Li02 13239 Li02
13240 \BCL2/BCL2_reg[4]_in Li03 13240 Li03
13241 \BCL2/BCL2_reg[3]_in Li04 13241 Li04
13245 DFF00 n00 13245 n00
10 \BCL2/BCL2_reg[7] Lo00 10 Lo00

12 \BCL2/BCL2_reg[5] Lo02 12 Lo02
13 \BCL2/BCL2_reg[4] Lo03 13 Lo03
14 \BCL2/BCL2_reg[3] Lo04 14 Lo04
:::keeping BCL2 concentration high

-26 \BAD/BADp_14_reg[7] Lo16 26 Lo16
-11465 DFF16 n16 11465 n16
:::elimiate half of BADp14 original concentration

-34 \BAD/BAD_cyto_reg[7] Lo24 34 Lo24
-35 \BAD/BAD_cyto_reg[6] Lo25 35 Lo25
-36 \BAD/BAD_cyto_reg[5] Lo26 36 Lo26
-37 \BAD/BAD_cyto_reg[4] Lo27 37 Lo27
-38 \BAD/BAD_cyto_reg[3] Lo28 38 Lo28
-12085 \BAD/BAD_cyto_reg[3]_in Li28 12085 Li28
-12084 \BAD/BAD_cyto_reg[4]_in Li27 12084 Li27
-12083 \BAD/BAD_cyto_reg[5]_in Li26 12083 Li26
-12082 \BAD/BAD_cyto_reg[6]_in Li25 12082 Li25
-12150 \BAD/BAD_cyto_reg[7]_in Li24 12150 Li24
:::keeping BAD_cyto concentration low

-13209 \BID/tBID_mito_reg[1]_in Li46 13209 Li46
-13208 \BID/tBID_mito_reg[2]_in Li45 13208 Li45
-13207 \BID/tBID_mito_reg[3]_in Li44 13207 Li44
-13218 DFF46 n46 13218 n46
-54 \BID/tBID_mito_reg[3] Lo44 54 Lo44
-55 \BID/tBID_mito_reg[2] Lo45 55 Lo45
-56 \BID/tBID_mito_reg[1] Lo46 56 Lo46
:::keeping tBID_mito concentration low

-72 \BID/tBID_cyto_reg[1] Lo62 72 Lo62
-11358 \BID/tBID_cyto_reg[1]_in Li62 11358 Li62
:::keeping tBID_cyto concentration low

4719 NOT_amtBAD_cyto[2] 4719
4953 NOT_amtBAD_cyto[3] 4953
:::keeping the translocation of BAD very low

-6572 NOT_amtCyto[4] 6572

-6768 NOT_amtCyto[5] 6768
-6999 NOT_amtCyto[6] 6999
-7146 NOT_amtCyto[7] 7146
:::keeping the translocation of tBID from cyto to mito very high

8862 NOT_wireCal[7] 8862
10872 NOT_amtCal[7] 10872
:::elimitate half of the flow of BADp14 to BAD_cyto

9656 NOT_tCyto_addv[1] 9656
:::keeping addition to tBID low

-11180 NOT_atBID[2] 11180
-11181 NOT_atBID[3] 11181
-11182 NOT_atBID[4] 11182
-11183 NOT_atBID[5] 11183
-11184 NOT_atBID[6] 11184
11037 atBID[7] po119 11037 po119
:::keeping reaction between BAD to break up tBID_BCL2 high, but only BAD is deducted.
:::This fault is located in the BAD module, so tBID_mito is not affected, depleting BAD

-11659 NOT_aBCL2[2] 11659
-11660 NOT_aBCL2[3] 11660
-11661 NOT_aBCL2[4] 11661
-11662 NOT_aBCL2[5] 11662
-11663 NOT_aBCL2[6] 11663
11607 aBCL2[7] po111 11607 po111
:::keeping BCL2 reaction with BAD high, hence depleting BAD

-12354 NOT_BCL2_inc[3] 12354
-12355 NOT_BCL2_inc[4] 12355
-12356 NOT_BCL2_inc[5] 12356
-12357 NOT_BCL2_inc[6] 12357
-12358 NOT_BCL2_inc[7] 12358
:::keeping BCL2 reaction with BAD_mito high, and BAD tBID_BCL2, but tBID is not affect
:::Hence depleting BAD

11426 NOT_Cyto_addv[3] 11426
11427 NOT_Cyto_addv[4] 11427

11428 NOT_Cyto_addv[5] 11428
11429 NOT_Cyto_addv[6] 11429
11430 NOT_Cyto_addv[7] 11430
:::keeping addition to BAD cyto low

-11745 NOT_atBCL2[2] 11745
-11746 NOT_atBCL2[3] 11746
-11747 NOT_atBCL2[4] 11747
-11748 NOT_atBCL2[5] 11748
-11749 NOT_atBCL2[6] 11749
-11750 NOT_atBCL2[7] 11750
:::keeping tBID reaction with BCL2 high

-12366 NOT_tMito_dc[2] 12366
-12367 NOT_tMito_dc[3] 12367
-12368 NOT_tMito_dc[4] 12368
-12369 NOT_tMito_dc[5] 12369
-12370 NOT_tMito_dc[6] 12370
-12414 NOT_tMito_dc[7] 12414
:::keeping tBID_mito deduction high

12069 NOT_tMito_addv[1] 12069
12070 NOT_tMito_addv[2] 12070
12071 NOT_tMito_addv[3] 12071
:::keeping tBID_mito addition low

B.3.2 Single Stuck at Faults For TN

Single stuck at fault for sequence TN
All these faults cause an non apoptotic sequence to become apoptotic (concentration of >=10)

13182 \BAD/BAD_mito_reg[7]_in Li08 13182 Li08
13183 \BAD/BAD_mito_reg[6]_in Li09 13183 Li09
13184 \BAD/BAD_mito_reg[5]_in Li10 13184 Li10
13185 \BAD/BAD_mito_reg[4]_in Li11 13185 Li11
13186 \BAD/BAD_mito_reg[3]_in Li12 13186 Li12
13187 \BAD/BAD_mito_reg[2]_in Li13 13187 Li13
13193 DFF08 n08 13193 n08
13194 DFF09 n09 13194 n09
13195 DFF10 n10 13195 n10

13196 DFF11 n11 13196 n11
13197 DFF12 n12 13197 n12
13198 DFF13 n13 13198 n13
18 \BAD/BAD_mito_reg[7] Lo08 18 Lo08
19 \BAD/BAD_mito_reg[6] Lo09 19 Lo09
20 \BAD/BAD_mito_reg[5] Lo10 20 Lo10
21 \BAD/BAD_mito_reg[4] Lo11 21 Lo11
22 \BAD/BAD_mito_reg[3] Lo12 22 Lo12
23 \BAD/BAD_mito_reg[2] Lo13 23 Lo13
:::Any of the BAD_mito reg s-a 1, break up tBID_BCL2 complex

12150 \BAD/BAD_cyto_reg[7]_in Li24 12150 Li24
12082 \BAD/BAD_cyto_reg[6]_in Li25 12082 Li25
12151 DFF24 n24 12151 n24
12089 DFF25 n25 12089 n25
34 \BAD/BAD_cyto_reg[7] Lo24 34 Lo24
35 \BAD/BAD_cyto_reg[6] Lo25 35 Lo25
:::Need BAD_cyto high bit to s-a to cause BID_mito to rise enough

13203 \BID/tBID_mito_reg[7]_in Li40 13203 Li40
13204 \BID/tBID_mito_reg[6]_in Li41 13204 Li41
13205 \BID/tBID_mito_reg[5]_in Li42 13205 Li42
13206 \BID/tBID_mito_reg[4]_in Li43 13206 Li43
13207 \BID/tBID_mito_reg[3]_in Li44 13207 Li44
13212 DFF40 n40 13212 n40
13213 DFF41 n41 13213 n41
13214 DFF42 n42 13214 n42
13215 DFF43 n43 13215 n43
13216 DFF44 n44 13216 n44
50 \BID/tBID_mito_reg[7] Lo40 50 Lo40
51 \BID/tBID_mito_reg[6] Lo41 51 Lo41
52 \BID/tBID_mito_reg[5] Lo42 52 Lo42
53 \BID/tBID_mito_reg[4] Lo43 53 Lo43
54 \BID/tBID_mito_reg[3] Lo44 54 Lo44
:::Any tBID_mito cause tBID_cyto to go high

11352 \BID/tBID_cyto_reg[7]_in Li56 11352 Li56
11353 \BID/tBID_cyto_reg[6]_in Li57 11353 Li57
11354 \BID/tBID_cyto_reg[5]_in Li58 11354 Li58
11355 \BID/tBID_cyto_reg[4]_in Li59 11355 Li59

11356 \BID/tBID_cyto_reg[3]_in Li60 11356 Li60
11360 DFF56 n56 11360 n56
11361 DFF57 n57 11361 n57
11362 DFF58 n58 11362 n58
11363 DFF59 n59 11363 n59
11364 DFF60 n60 11364 n60
66 \BID/tBID_cyto_reg[7] Lo56 66 Lo56
67 \BID/tBID_cyto_reg[6] Lo57 67 Lo57
68 \BID/tBID_cyto_reg[5] Lo58 68 Lo58
69 \BID/tBID_cyto_reg[4] Lo59 69 Lo59
70 \BID/tBID_cyto_reg[3] Lo60 70 Lo60
:::tBID_cyto concentration stuck at high

-4592 NOT_amtBAD_cyto[1] 4592
-4719 NOT_amtBAD_cyto[2] 4719
-4953 NOT_amtBAD_cyto[3] 4953
4596 amtBAD_cyto[4] po132 4596 po132
:::Translocation of BAD s-a even only bit 1

-6790 NOT_Mito_addv[3] 6790
-6791 NOT_Mito_addv[4] 6791
-6792 NOT_Mito_addv[5] 6792
-6841 NOT_Mito_addv[6] 6841
-6908 NOT_Mito_addv[7] 6908
:::BAD mito addition increase, translocation increase

-6807 NOT_aMito[2] 6807
-6808 NOT_aMito[3] 6808
-6809 NOT_aMito[4] 6809
-6810 NOT_aMito[5] 6810
6664 aMito[6] po038 6664 po038
:::amount leaving tBID mito s-a, create excess tBID_cyto

-6820 NOT_wireCal[1] 6820
-6959 NOT_wireCal[2] 6959
-7364 NOT_wireCal[3] 7364
-7689 NOT_wireCal[4] 7689
-8081 NOT_wireCal[5] 8081
-8433 NOT_wireCal[6] 8433
-8862 NOT_wireCal[7] 8862

-10077 NOT_amtCal[1] 10077
-10329 NOT_amtCal[2] 10329
-10422 NOT_amtCal[3] 10422
-10479 NOT_amtCal[4] 10479
-10645 NOT_amtCal[5] 10645
-10816 NOT_amtCal[6] 10816
-10872 NOT_amtCal[7] 10872
:::amount cal react to keep plenty of BAD_cyto supply

-9658 NOT_tCyto_addv[3] 9658
-9659 NOT_tCyto_addv[4] 9659
-9660 NOT_tCyto_addv[5] 9660
-9661 NOT_tCyto_addv[6] 9661
-9733 NOT_tCyto_addv[7] 9733
:::tBID cyto supply increase

-11156 NOT_aBAD[1] 11156
-11157 NOT_aBAD[2] 11157
-11158 NOT_aBAD[3] 11158
-11159 NOT_aBAD[4] 11159
-11160 NOT_aBAD[5] 11160
-11161 NOT_aBAD[6] 11161
11010 aBAD[7] po023 11010 po023
:::BAD release tBID_BCL2

-11429 NOT_Cyto_addv[6] 11429
-11430 NOT_Cyto_addv[7] 11430
:::BAD cyto increase

-12071 NOT_tMito_addv[3] 12071
-12072 NOT_tMito_addv[4] 12072
-12073 NOT_tMito_addv[5] 12073
-12074 NOT_tMito_addv[6] 12074
-12075 NOT_tMito_addv[7] 12075
:::tBID mito addition increase

-12475 NOT_Mito_dc[3] 12475
-12476 NOT_Mito_dc[4] 12476
-12477 NOT_Mito_dc[5] 12477

-12478 NOT_Mito_dc[6] 12478
-12479 NOT_Mito_dc[7] 12479
:::BCL2 deduction increase, depleting BCL2

B.3.3 Double Stuck at Faults For TA

Double stuck at fault for sequence TA

F: -50 -5748 0000000100000001000000000000111011101011001010011000101100000000(1, 1, 0, 14, 235, 41, 139, 0)
F: -5748 12075 0000000100000001000000000000111011101011001010111000101100000000(1, 1, 0, 14, 235, 43, 139, 0)
F: -5748 -13203 0000000100000001000000000000111011101011001010011000101100000000(1, 1, 0, 14, 235, 41, 139, 0)

-50 -5748
\BID/tBID_mito_reg[7] Lo40 50 Lo40
NOT_amtCyto[1] 5748

---

tBID mito 7 s-a 0, there is a leak (half) in mito, by itself it doesn't cause error since bit 7 is not used in normal case amtCyto 1 s-a 1, puming in tBID-mito extra stuff by using amtCyto, by itself, it create excess tBID-mito at time get tBID_cyto up to 12 but no much more When combine, tBID-mito reach 126 then 129 but mito bit is s-a 0 so tBID drop down to 1,
Effect: having extra tBID-mito that rollover
In fact it did reach apoptotic state briefly before rolling over.

-5748 12075
NOT_amtCyto[1] 5748
NOT_tMito_addv[7] 12075

---

tBID_add s-a 0, amount adding to tBID mito is leaking at 7, not even used, similar effect as if tBID_mit_reg s-a. amtCyto 1 s-a 1, puming in tBID-mito extra stuff by using amtCyto, by itself, it create excess tBID-mito at time get tBID_cyto up to 12 but no much more. When combine, rollover

-5748 -13203
NOT_amtCyto[1] 5748
\BID/tBID_mito_reg[7]_in Li40 13203 Li40

Same as 1

B.3.4 Double Stuck at Faults For TN

Double stuck at fault for sequence TN
Double Faults that cause apoptosis when it is not suppose to

39 4318
\BAD/BAD_cyto_reg[2] Lo29 39 Lo29
NOT_amtBAD_cyto[0] 4318

4318 12086
NOT_amtBAD_cyto[0] 4318
\BAD/BAD_cyto_reg[2]_in Li29 12086 Li29

4318 12093
NOT_amtBAD_cyto[0] 4318
DFF29 n29 12093 n29

BADcyto 2 register s-a 1, having more BAD_cyto increase tBID_mito to 8 instead of 6 help in some case, but end of having not enough BAD-cyto after depelting it
amtBAD_cyto 0 s-a 0 actually, only transfer 2 unit at a time, keeping BAD_cyto higher hence tBID_mito concentration remain low , only 2 instead of 6, but more BAD_cyto remain unused tBID_mito not high enough
When combine, having extra BAD_cyto and prolonging transfer, avoid transfering 1 unit but transfering 2 unit with extra BAD_cyto, 39 alone, kept tranfering 1 unit and eventually just not having not strong enough current to transfer anymore having 4318, keep high concentration, larger pressure hence higher velocity since when BAD_cyto is at 40, the s-a 1 for bit 1 help, but when BAD_cyto is 14, s-a does not help
Effect: creation of BAD_cyto a bit higher with a bit smaller tranlocation
Think about a container with a valve, if the valve is wide open, and there is a constant supply, the incoming supply will just go right out, the valve is undercapcity, if valve is half closed, let the tank accumulate and push out higher pressure with the supply it make it last longer

-11660 12354
NOT_aBCL2[3-7] 11660
NOT_BCL2_inc[3-7] 12354

---

aBCL2 3 s-a 1, is one component of BCL_inc, causing excess movement of BAD_mito, depleting it and BCL2, there is no BAD_mito left to go create tBID_mito
BCL_inc 3-7 s-a 0, amount going to BAD_BCL2 and going away from BAD_mito. Bit 3-7 is not even used in normal case, only increment by 2

When combine, BCL_inc block the aBCL2 s-a, but aBCL2 s-a still depleting BCL2 Because of the block BAD_mito is not being depleted
Effect, BCL2 is depleted early leaving all the BAD_mito to cause apoptosis

-11746 12367
NOT_atBCL2[3-7] 11746
NOT_tMito_dc[3-7] 12367

---

atBCL2 s-a 1 amount of BCL2 reaction is boosted (a component of Mito_dc) depleting BCL2 and tBID_mito
tMito_dc s-a 0 amount of tBID_mito deduction is lessen 3-7 is not even used in normal case, tBID_mito is only 6 at the most
When combine, Mito_dc block the atBCL2 s-a, so BCL2 is depleted while tBID_mito is not being deplete

11743 12365
NOT_atBCL2[0] 11743
NOT_tMito_dc[1] 12365

---

atBCL2 0 s-a 0, cause a unit of 2, keeping BCL2 and tBID_mito concentration higher, prolonging it. wiretBCL2 is higher but only unit of 2 is used up
tMito_dc 1 s-a 0, amount of tBID_mito deduction is lessen, in normal cases, aMito is zero, only aBCL2 contribute, and tMito_dc only use bit 0
atBCL2 is amout to subtract from BCL2
tMito_dc is amount to subtract from tBID_mito
when combine, atBCL2 keep a unit of 2 of tBID_mito reaction BCL2 in turn keeping tBID_mito concetnration higher. Because itis kept higher, tMito_dc s-a has an effect. This s-a kept deduction of tBID_mito lower; however BCL2 is still deduced the full amount.
Effect: cause the depletion of BCL2 twice as much as tBID_mito, changing the nature of reaction ratio

6604 12365
NOT_wiretBCL2[0] 6604
NOT_tMito_dc[1] 12365

---

wiretBCL2 0 s-a 0, tMito_dc 1 s-a 0
same effect as above

Vita

Huy H. Lam was born in Ho Chi Minh City, Vietnam. On November 1997, he moved to the U.S.A. along with his family. He attended Virginia Tech in Fall 2002 and obtained his Bachelor of Science in Computer Engineering in Spring 2006. He continued his Masters in Computer Engineering and joined Dr. Hsiao and his research group in Fall 2006. Since then, his research has been involved with verification techniques for the discrete transition system and fault model analysis. His works focus on applying hardware testing and verification approaches to solve biological systems, specifically, a subsection of apoptosis signaling pathways.

Permanent Address: 14136 Darkwood circle
Centreville, VA 20121.