Chapter 14
Test Generation for Detection of Malicious Parametric Variations

Yuanwen Huang and Prabhat Mishra

14.1 Introduction

Third-party IPs are widely used in SoC design methodology. Some of these IPs may come from untrusted third-party vendors. It is crucial to ensure that an IP block is not vulnerable to input conditions that violate its non-functional (parametric) constraints, such as power, temperature, or performance. Power supply voltages, increased integration densities, and higher operating frequencies, among other factors, are producing devices that are more sensitive to power dissipation and reliability problems.

Figure 14.1 shows a scenario where an adversary can construct a specific test (input sequence) that can maximize the pear power or the peak temperature. We use the terms *power virus* and *temperature virus* to refer to the tests that can violate the peak power and peak temperature, respectively. Excessive power dissipation can lead to overheating, electromigration, and a reduced chip lifetime. Moreover, large instantaneous power consumption causes voltage drop and ground bounce, resulting in circuit delays and soft errors. Therefore, accurate power estimation during the design phase is crucial to avoid a time-consuming re-design process and in the worst-case an extremely costly tape-out failure. As a result, reliability analysis has steadily become a critical part of the design process of digital circuits.

Figure 14.2 shows different types of power and temperature viruses for different IPs. For gate-level or RTL-level IPs, a power virus is a set of test vectors that can create excessive instantaneous power consumption. For a processor IP, a power virus

Y. Huang (✉) • P. Mishra
Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL, USA
e-mail: yuanwenhuang@ufl.edu; prabhat@ufl.edu

P. Mishra et al. (eds.), *Hardware IP Security and Trust*,
DOI 10.1007/978-3-319-49025-0_14

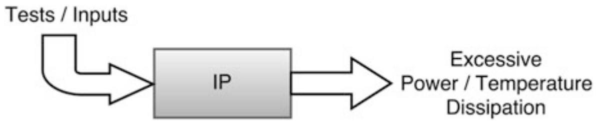

Fig. 14.1 Malicious inputs can cause excessive power/temperature dissipation

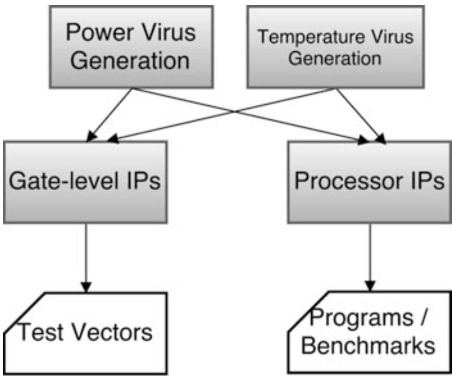

Fig. 14.2 Different types of power viruses

is a program that can cause peak power during its execution. Similarly, temperature virus generation can be for gate-level IPs or processor IPs. The reminder of this chapter describes these four test generation scenarios in detail.

14.2 Power Virus for Gate-Level IPs

The basic idea is to generate a power virus that can maximize the switching activity in the IP block. In case the peak power violates the threshold, the designers may need to re-design the IP block to reduce the peak power. In CMOS circuits, power dissipation depends on the extent of circuit switching activity, which is input-pattern dependent. The instantaneous power dissipation due to two consecutive input binary vectors is proportional to:

$$P \propto \sum_{\text{all gates}} T(g) * C(g) \tag{14.1}$$

where $C(g)$ denotes the output capacitance of gate g, and $T(g)$ indicates whether gate g switches or not when the circuit is fed with the two input vectors. $T(g)$ is 1 if the gate switches, and it is 0 if the gate does not switch. P is an estimation of the total power consumption.

The maximum circuit activity estimation problem aims at finding the input patterns which cause peak instantaneous dynamic power, a worst-case scenario where excessive simultaneous gate switching imposes extreme current demands on the power grid, leading to unwanted voltage drops. One naive approach is to exhaustively search for two consecutive binary input vectors which can induce the maximum number of switching. Unfortunately, this problem is NP-complete for combinational as well as sequential circuits. The time complexity for this exhaustive search is $O(4^n)$, where n is the number of primary input pairs of the circuit and each input signal has four possible pairs of values (i.e., 0-0, 0-1, 1-1, 1-0).

Existing methods for maximum power estimation can be classified into two broad categories: simulation based and non-simulative approaches. For a circuit with large number of primary inputs, it is not possible to exhaustively search all input patterns for maximum power consumption. The only practical way to solve the problem is to generate a tight lower bound for maximum attainable power. During circuit simulation, Monte Carlo based technique can be used to estimate the maximum power [3, 22]. By estimating the mean and deviation of power and monitor the maximum power during simulation, a lower bound of the peak power can be measured from a statistical point of view. Non-simulative approaches use characteristics of the circuit and stochastic properties of input vectors to perform power estimation without explicit circuit simulation.

In the following subsections, we will explain in detail a few non-simulative methods for combinational circuits, including a pseudo-Boolean satisfiability approach and three other approaches based on Automatic Test Pattern Generation (ATPG). Then we will show how to extend the methods from combinational circuits to sequential circuits.

14.2.1 Pseudo-Boolean Satisfiability Approach

In CMOS circuits, the capacitance load of a gate output is proportional to the number of fanout. Hence, the power dissipation of two consecutive input vectors x^1 and x^2 can be rewritten as:

$$\sum_{i=1}^{m} F_i \cdot (g_i(x^1) \oplus g_i(x^2)) \tag{14.2}$$

where $g_i(*)$ denotes the Boolean function output of gate g_i with the specified input vector, and F_i denotes the fanout factor of gate g_i. This equation is an estimation of the total power consumption of the circuit, and it is the objective function to be maximized.

The pseudo-Boolean satisfiability approach was proposed in [14]. Figure 14.3b shows the new circuit N that is constructed for the pseudo-Boolean SAT problem. The new circuit N consists of two replicas of the original circuit T, namely T^1 and T^2. The primary input vectors (x^1 and x^2) are applied to the two replicas (T^1 and T^2), respectively. For every pair of corresponding gates, for example, g_1^1 in T^1 and g_1^2 in T^2, a new XOR gate xor_1 is constructed with g_1^1 and g_1^2 as inputs. The output of each XOR gate xor_i yields $g_i(x^1) \oplus g_i(x^2)$.

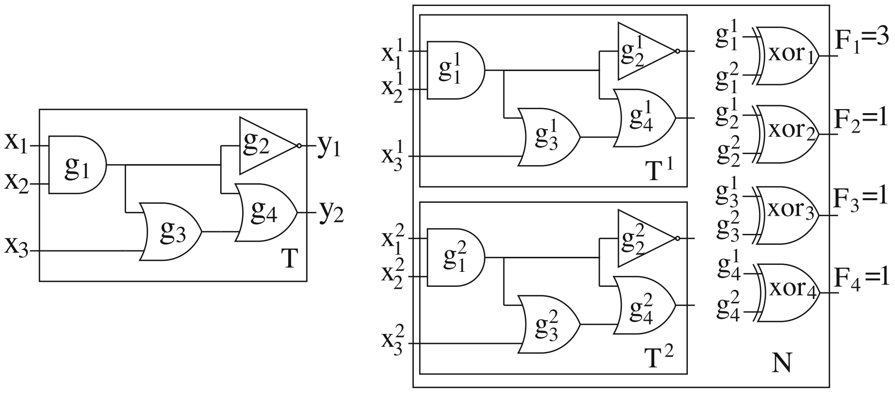

Fig. 14.3 Pseudo-Boolean optimization (PBO) formulation for combinational circuits [14]. (**a**) The original circuit. (**b**) The new circuit

The pseudo-Boolean SAT problem can be formed to maximize P as follows:

$$\begin{aligned} P &= \sum_{i=1}^{m} F_i \cdot \text{xor}_i \\ \text{subject to} \quad \Psi &= \text{CNF}(N) \end{aligned} \tag{14.3}$$

The objective function P is the total number of switches in the original circuit. CNF(N) is the *Conjunctive Normal Form* of the new circuit N, which is the constraints that the pseudo-Boolean problem needs to satisfy.

The problem in Eq. (14.3) can be solved using pseudo-Boolean optimization (PBO) solvers. However, as a formal method, this approach is not scalable due to the complexity of pseudo-Boolean SAT problem for large circuits.

14.2.2 Largest Fanout First Approach

Instead of directly searching for input vector pairs (x^1, x^2) to maximize P, the Largest Fanout First approach [22] assigns transitions to the internal gates in a greedy way as shown in Fig. 14.4. The gates are sorted by the fanout number in decreasing order. In every iteration, a gate g_i which is not tried and has the largest fanout is selected to assign a transition: $g_i(x^1) \oplus g_i(x^2) = 1$. The assignment of gate g_i is justified by two processes: *backtracking* (backward to inputs) and *implication* (forward to outputs) in the circuit. If the justification of the transition assignment fails, i.e., conflicts happen during justification, the node values of the circuit will be

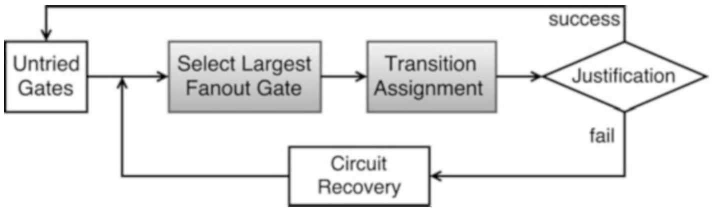

Fig. 14.4 Largest fanout first approach [22]

recovered, which means the newly implied values will be reset to previous values. The algorithm tries gates one by one to assign transitions until all gates have been processed. Interested readers can refer to [22] for details of the justification process.

14.2.3 Cost-Benefit Analysis Approach

The Largest Fanout First (LFF) approach [22] is a greedy algorithm that assigns transitions to high-fanout gates first. However, LFF neglects an important fact that when a specific gate (node) is marked for switching, it does create certain number of switching events but it may restrict some other switching in the future. To address the flaw in LFF , the cost-benefit approach [7] suggests that every assignment should have a cost-benefit analysis. After applying a certain assignment to a gate, there might be changes in future switching possibility for all other untried gates. The cost-benefit analysis on all possible assignments selects the most favorable assignment, which facilitates the overall optimization process to enable more efficient power virus generation. The *switching probability* of a gate and the *cost-benefit* function for an assignment are defined as follows:

$$\begin{aligned} \mathrm{SP}(g_i) &= \frac{\text{number of fanin combinations that make } g_i \text{ switch}}{\text{total number of available fanin combinations}} \\ \mathrm{CB}(a) &= \sum_{\text{untried gates}} \Delta\mathrm{SP}(g_i) * F_i \end{aligned} \tag{14.4}$$

where a is the assignment made at the current iteration and F_i is the fanout number of gate g_i. $\mathrm{SP}(g_i)$ is the switching probability of gate g_i, which is the ratio of the number of fanin combinations that can make g_i switch to the total number of available fanin combinations. $\Delta\mathrm{SP}(g_i)$ is the difference in switching probability after and before the assignment. If $\Delta\mathrm{SP}(g_i)$ is positive, this assignment increases the switching probability of gate g_i. In this case, $\Delta\mathrm{SP}(g_i) * F_i$ is considered as the benefit because of this assignment; otherwise, it is considered as the cost. $\mathrm{CB}(a)$ is the sum of benefit/cost of all untried gates. The assignment with the largest $\mathrm{CB}(a)$ is the most beneficial one, and it is most favorable for untried gates to switching in the future.

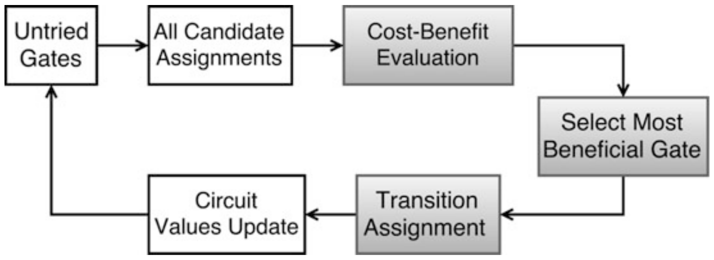

Fig. 14.5 Cost-benefit analysis approach [7]

The workflow for the cost-benefit approach is shown in Fig. 14.5. At the beginning of each iteration, the cost-benefit analysis is done for all possible assignments of all untried gates. The cost-benefit analysis also involves justification of the assignment throughout the circuit. Only feasible assignments are evaluated for cost-benefit values. The gate with the most beneficial assignment is selected, and the assignment is applied to the circuit to update values in the circuit. The algorithm stops when all gates have been tried. Interested readers can refer to [7] for details.

14.2.4 Power Virus for Sequential Circuits

So far, we have described how to construct power virus for combinational circuits. The estimation of power consumption for sequential circuits [9, 10, 16, 17] is a natural extension of the above approaches. The dynamic power of the combinational portion in a sequential circuit is still the same as shown in Eq. (14.1). The problem of single-cycle peak power in a sequential circuit is very similar to that of instantaneous power in combinational circuit. The only difference is that sequential circuits have state elements (flip-flops or storage elements). Let S_i denote the state of flip-flops at the beginning of the *i-th* clock cycle. Let I_i denote the primary input vector for the *i-th* clock cycle. The sequence $(S_1, I_1, S_2, I_2, \ldots, I_n, S_{n+1})$ represents the change of states of the circuit with a set of input vectors.

The approaches discussed above for power virus generation for combinational circuits can be extended to sequential circuits for single-cycle peak power generation. The outputs of one clock cycle depend on both the input vector and the state of flip-flops from previous cycle. To apply the above approaches, we need to initialize the state of flip-flops to a known state. If the circuit is full-scan design, the state of the flip-flops can be initialized to any arbitrary value. For circuits without a full scan chain inserted, the initialization can be done by *warming up* the circuit as discussed in [10, 17]. The initial state of flip-flops are assumed to be undefined and input vectors are generated to feed the circuit until all the flip-flops hold a definite value. In the case that the initial state of a circuit is not fully controllable, the above procedure cannot initialize some of the flip-flops. As stated in [10], random values can be speculated to these flip-flops. Once the state of the circuit is defined, we can apply the approaches for power virus generation for combination circuits, which is to find a pair of input vectors for maximum power consumption. In Sect. 14.4, we will discuss more about *peak k-cycle power* and *peak sustainable power*.

14.3 Power Virus for Processor IPs

A power virus in an architecture-level IP, or a processor, is a computer program that can execute specific machine code to have excessive CPU power consumption. Since the cooling system of a computer is designed with the budget of thermal design power, rather than the maximum power, a power virus could cause the system to overheat. If the power management logic cannot stop the processor in time, the power virus may permanently damage the system. In order to design a suitable power budget and various power management features, it is crucial to understand the power characteristics of the system and get accurate estimation of the attainable worst-case power dissipation. It is important to note that power virus for processor is a special case of power virus discussed above.

14.3.1 Stress Benchmarks

Power virus programs, or stress benchmarks, are often used for thermal testing of computer components during the design phase. Thermal testing is part of the stability testing when designing and benchmarking a reliable computer system. MPrime [15] and CPUburn-in [2] are the most popular benchmarks to stress-test a CPU.

MPrime [15] is a freeware application that searches for Mersenne prime numbers. It is called the torture test which has been extremely popular among PC enthusiasts and overclockers as a stability testing utility. The stress-test feature can be configured by setting the size for fast Fourier transform during its primality check. The program can subject the processor and memory to an incredibly intense workload, and it halts when it encounters even one minor error. The amount of time that a processor remains successfully stable while running MPrime is used as a measure of the system's stability. This feature has made MPrime suitable to test the stability of processor and memory, as well as the cooling efficiencies.

CPUburn-in [2] is a stability testing tool for overclockers, written by Michal Mienik. The program heats up any x86 processor to the maximum possible operating temperature. The program constantly runs FPU intensive functions for a user specified period of time, and it continuously monitors for errors ensuring that the CPU does not generate errors under overclocking conditions.

The torture tests such as MPrime [15] and CPUburn-in [2] can put heavy workload on the processor and push the system towards high power dissipation.

But there is no guarantee that the worst-case power dissipation can be attained by these benchmarks. Chip designers have to write hand-crafted power virus programs for a good estimation of the attainable peak power. However, it is very tedious and inefficient to manually design effective power virus for a given architecture. Moreover, power virus designed for one specific architecture usually cannot be directly used for a different processor IP. An automatic exploration and code generation methodology is needed to generate power viruses for architecture-level IPs. The following two subsections will introduce such an automatic approach which can generate power viruses for any architecture using genetic algorithm.

14.3.2 Power Virus Generation for Single-Core IPs

The goal is to generate a power virus program that can create the maximum worst-case power consumption. It is important to note that the worst-case power of a processor is not simply the sum of the maximum power of all components. It is almost impossible to have all components/resources achieve the maximum utilization at the same time. Due to the stalls in processor pipelines and contention of other shared resources, such as cache or memory ports, the peak total power is significantly smaller than the sum of peak power of all components.

When an instruction goes through different pipeline stages (fetch, decode, execute, memory, and commit), it will activate certain functional units and components in the processor. The power consumed by one instruction depends on three factors: (1) the opcodes and operands of the instruction, (2) the other instructions that are competing resources with this instruction, (3) the hardware constraints of the architecture. A power virus consists of a sequence of instructions that would create maximum power in a given architecture.

It is crucial that if the power virus can evolve and adapt to the hardware configurations of the processor IP, so as to best utilize all the components at all stages of the pipeline. Machine learning and genetic algorithms can be used to drive this process to search for power viruses with maximum power consumption. Such a learning approach (such as [12] and [5]) is not constrained by the hardware configurations, thus is applicable to different processor IPs.

The framework for automatic generation of synthetic power virus programs is shown in Fig. 14.6. The *Genetic Algorithm* generates the parameter values for the potential candidates for the power viruses as it searches through the parameter space. These abstract program parameters are fed to the *Code Generator* that generates a synthetic C program containing embedded assembly instructions based on these specified parameters. The C programs will be compiled into binary programs, which are candidates for power viruses. The simulator executes each binary program and gets the estimated maximum power consumption. The power consumption for each binary program will be the fitness value, which indicates the fitness of the set of program parameters used to generate this program. The *Genetic Algorithm* will take all the fitness values as feedback, tune the parameters to intelligently search for next

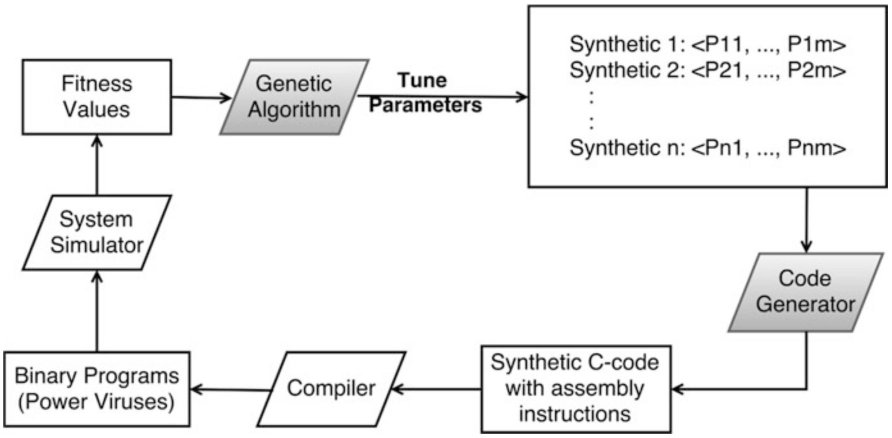
Fitness Values
Genetic Algorithm
Tune Parameters
Synthetic 1: <P11, ..., P1m>
Synthetic 2: <P21, ..., P2m>
Synthetic n: <Pn1, ..., Pnm>
System Simulator
Code Generator
Binary Programs (Power Viruses)
Compiler
Synthetic C-code with assembly instructions

Fig. 14.6 A framework to construct synthetic benchmarks (power virus) using genetic algorithm [5]

Table 14.1 Search space and parameter settings [5]

Category	Program parameter	Range
Control flow predictability	Number of basic blocks	(10–200)
	Average block size	(10–100)
	Average branch predictability	(0.8–1.0)
Instruction mix weights	Integer instructions (ALU, mul, div)	(0–4, 0–4, 0–4)
	FP instructions (add, mul, div)	(0–4, 0–4, 0–4)
	Load/store instructions (ld, st)	(0–4, 0–4)
Instruction level parallelism	Register dependency distance distribution	
Data locality	Stride value distribution for load/store	
	Data footprint	(1–200)
Memory level parallelism	Mean of memory level parallelism	(1–6)

generation of parameters which can have even higher power consumption (fitness value). The process iteratively continues until the genetic algorithm converges to find the best power virus for a given system configuration. The program parameter space is presented in Table 14.1 in Sect. 14.3.2.1. The process of *Code Generator* is explained in detail in Sect. 14.3.2.2.

14.3.2.1 Exploration Space of Program Characteristics

Given a specific architecture, we would like to adaptively change the parameters of the power virus program to maximize power consumption. The parameters of the program include control flow predictability, instruction mix parameters, instruction level parallelism, data locality, and memory level parallelism. The exploration space of all these parameters is shown in Table 14.1.

The control flow predictability of the program is set by three different parameters: the number of basic blocks, the average block size, and the average branch predictability. The number of basic blocks and block size defines the instruction footprint of the program. The branch predictability of a program is important as it affects the overall throughput of the pipeline. When a misprediction happens, the pipeline has to be flushed and this would result in reduced activity in the pipeline. The percentage of correctly predicted branches dictates the activity level of the branch predictor. The power consumption of a branch predictor is usually around 5 % of the overall processor power.

The instruction mix determines the frequency of each type of instruction in the program. Let us consider eight types of instructions (three types of integer instructions, three types for FP instructions and load/store instructions) in Table 14.1. Since different instructions have various latencies and power consumption, the instruction mix has a major impact on the overall power consumption. The typical power consumption of the integer and floating point ALU is around 4–6 % and 6–12 %, respectively.

The instruction level parallelism is determined by the register dependency distance distribution. The register distance between instructions can be (1, 2, 4, 8, 14, 16, 20, 32, 48, 64) and the program will search for the best distribution in these ten bins. This parameter impacts the throughput of the pipeline directly. The overall throughput of the pipeline is related to the power consumption of the instruction window, clock, and rename logic. The typical power consumption for the instruction window and the clock subsystem is around 8 % and 20 %, respectively.

The data level parallelism is determined by data locality and memory level parallelism. Data locality includes two aspects: the stride value between load/store instructions, the data footprint (the number of iterations before resetting to the beginning of the data array). The stride value between the addresses of load/store instructions can be (0, 4, 8, 12, 16, 32, 64) and the program will search for the best distribution. Data footprint controls the number of cache lines that will be touched by load/store instructions. The average number of long-latency outstanding loads (loads with big strides that will cause miss in the last level cache) represents the memory level parallelism. Interested readers can refer to [5] for details.

14.3.2.2 Code Generation

The process of code generation is illustrated in Fig. 14.7. First, the number of basic blocks in the synthetic program is fixed. For each basic block, the instruction type for every instruction using the instruction mix is selected. The basic blocks are then bound together using conditional jumps. For each instruction, one needs to find a producer instruction to assign a register dependency. The destination registers are assigned by round-robin. The source registers are assigned based on dependency. Memory access is modelled by having load/store accesses to a set of 1-D arrays

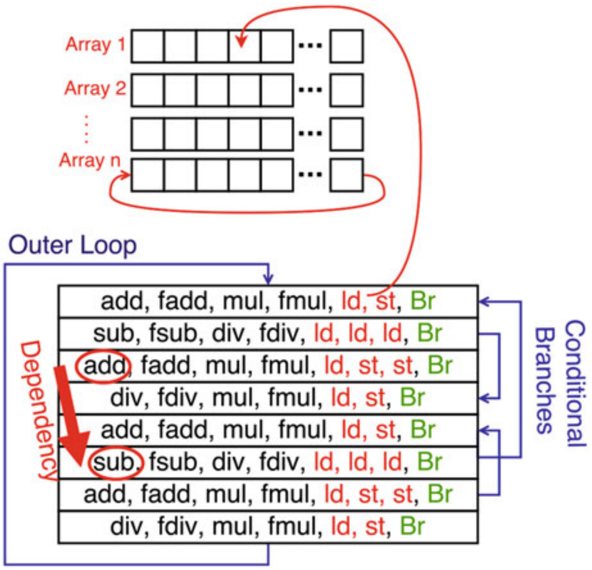
Array 1
Array 2
Array n
Outer Loop
add, fadd, mul, fmul, ld, st, Br
sub, fsub, div, fdiv, ld, ld, ld, Br
add, fadd, mul, fmul, ld, st, st, Br
div, fdiv, mul, fmul, ld, st, Br
add, fadd, mul, fmul, ld, st, Br
sub, fsub, div, fdiv, ld, ld, ld, Br
add, fadd, mul, fmul, ld, st, st, Br
div, fdiv, mul, fmul, ld, st, Br
Dependency
Conditional Branches

Fig. 14.7 Example of code generation of a synthetic power virus program [5]

in a stride fashion. Load/store instructions are grouped into pools and assigned to different arrays. The pointer of arrays are reset to the beginning of array when required data footprint is touched. Interested readers can refer to [5] for details.

14.3.3 Power Virus for Multi-Core IPs

For a multi-core IP, a power virus would be a multi-threaded program that has maximum power consumption. It is more challenging than single-core IP because of components like the interconnection network, shard caches, DRAM, and coherence directory, which also contribute significantly to the power consumption of a multi-core parallel system. We need to exploit these features of multi-core systems to the right extent. The parameter exploration space should take these features into consideration and use genetic algorithm to search for optimal parameters [1, 4, 13].

14.3.3.1 Space Exploration of Program Characteristics

Table 14.2 shows the program parameters for multi-threaded power virus generation. The first two categories (parallelism and shared data access) are specifically for multi-thread exploration, and the rest of the parameters are the same as in Table 14.1. The number of threads controls the amount of thread level parallelism of the synthetic program. The thread class and process assignment parameter choose various patterns which decides the assignments of threads to cores that they are

Table 14.2 Search space and parameter settings for multi-core IP [4]

Category	Program parameter	Range
Parallelism	Number of threads	(1–32)
Shared data access	Thread class and processor assignment	
	Percent memory accesses to shared data	(10–90)
	Shared memory access strides	(0–64)
	Coupled load-stores	(True/false)
Control flow predictability	Number of basic blocks	(10–200)
	Average block size	(10–100)
	Average branch predictability	(0.8–1.0)
Instruction mix weights	Integer instructions (ALU, mul, div)	(0–4, 0–4, 0–4)
	FP instructions (add, mul, div)	(0–4, 0–4, 0–4)
	Load/store instructions (ld, st)	(0–4, 0–4)
Instruction level parallelism	Register dependency distance distribution	
Data locality	Stride value distribution for load/store	
	Data footprint	(1–200)
Memory level parallelism	Mean of memory level parallelism	(1–6)

bound to execute. The percentage of memory accesses to shared data and the shared memory access strides will influence accesses to shared data. A coupled load-store is a load with a following store, which mimics a migratory sharing pattern of access. The coupled load-store parameter can be either set or unset.

14.3.3.2 Multi-Threaded Power Virus Generation

Figure 14.8 shows the flowchart of automatic generation of multi-threaded power viruses using genetic algorithm. The work flow is very similar to the power virus generation for single core, except that there are more parameters related to the multi-thread characteristics. Each set of parameters contain specification for the multi-thread characteristics as well as for each thread. Interested readers can refer to [4] for details.

14.4 Temperature Virus

14.4.1 Temperature Virus for Gate-Level IPs

Temperature virus can be built on the idea of power virus, because sustained high power could produce peak temperature. The sequence $(S_1, I_1, S_2, I_2, \ldots, I_n, S_{n+1})$ represents the change of states of the circuit with a set of input vectors. Figure 14.9 shows the definitions of *peak single-cycle power*, *peak n-cycle power*, and *peak*

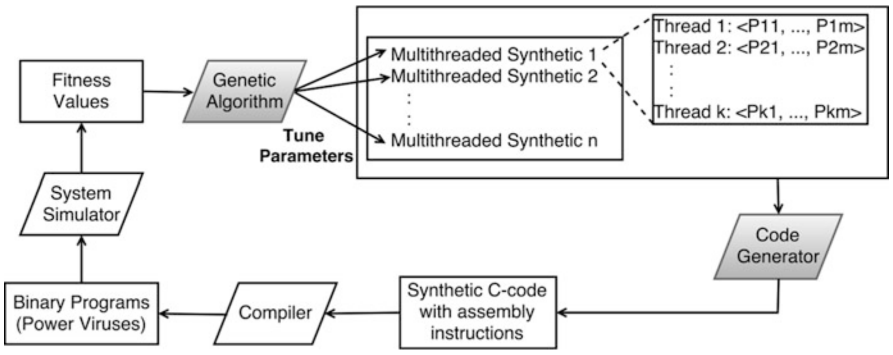
Fitness Values
Genetic Algorithm
Tune Parameters
Multithreaded Synthetic 1
Multithreaded Synthetic 2
Multithreaded Synthetic n
Thread 1: <P11, ..., P1m>
Thread 2: <P21, ..., P2m>
Thread k: <Pk1, ..., Pkm>
Code Generator
Synthetic C-code with assembly instructions
Compiler
Binary Programs (Power Viruses)
System Simulator

Fig. 14.8 A framework for multi-threaded power virus generation using genetic algorithm [4]

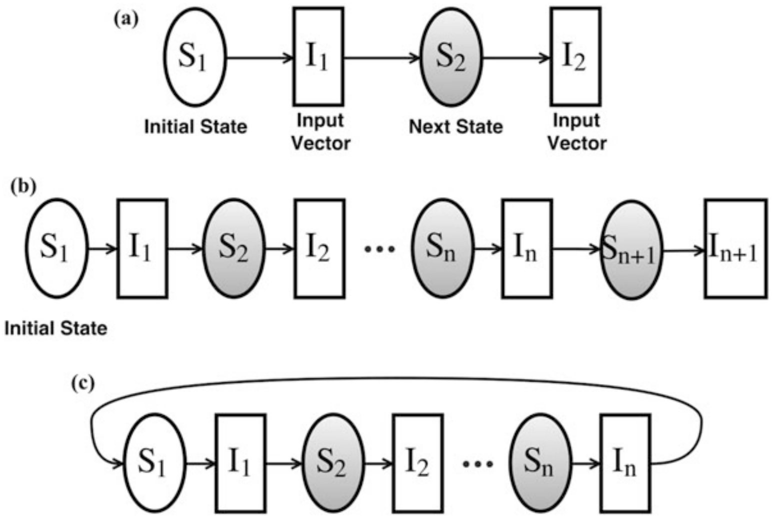

Fig. 14.9 Definitions of peak power measures. (**a**) Peak single-cycle power. (**b**) Peak *n*-cycle power (power measured and averaged over *n* cycles). (**c**) Peak sustainable power (power measured and averaged over *n* cycles, circuit returns initial state for every *n* cycles) [8]

sustainable power. For *peak single-cycle power*, the power virus would be a tuple (S_1, I_1, I_2) that has the maximum power consumption during one clock cycle. For *peak n-cycle power*, the power virus would be a sequence $(S_1, I_1, I_2, \ldots, I_n, I_{n+1})$ that has the maximum average power consumption over n clock cycles. For *peak sustainable power*, the power virus can make the circuit return to the initial state S_1 after n clock cycle. The power virus which can produce the *peak sustainable power* is actually a temperature virus, because we can repeatedly apply the input sequence $(I_1, I_2, \ldots, I_n)$ to have the circuit indefinitely remain on high power consumption.

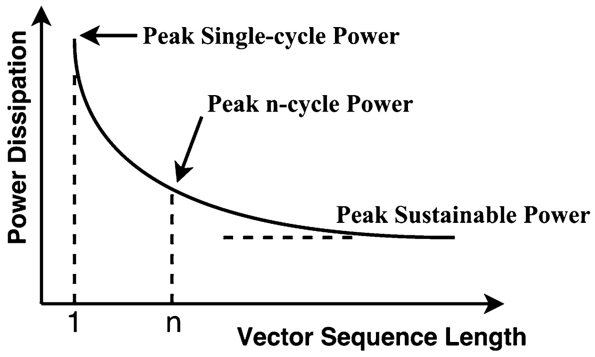

Fig. 14.10 Lower bound for peak power dissipation [8]

The initial state S_1 is important in determining the peak power/temperature of the sequential circuit. If the circuit state is fully controllable, the approaches for power virus generation discussed in Sect. 14.2 can be extended for *peak single-cycle power* and *peak n-cycle power* estimation. Genetic algorithm can also be used to generate the sequence $(S_1, I_1, I_2, \ldots, I_n, I_{n+1})$, with the initial state S_1 chosen from a pool of reachable and controllable states.

For *peak sustainable power* (peak temperature), the problem is to derive a long sequence of vectors that have high power dissipation which can be sustained. Hsiao et al. [8] have shown a typical trend for *peak n-cycle power* dissipation and the process of searching for *peak sustainable power*. As shown in Fig. 14.10, when n equals to 1, it is the *peak single-cycle power*. As n increases, the *peak n-cycle power* is expected to decrease if the vector sequence cannot sustain the power dissipation. The peak power levels off when n is large enough or a loop is found in the state transitions. There are two approaches to search for power virus with peak sustainable power. The first one is to start with a *peak n-cycle power* sequence, and then search for vectors to close the loop with as few state transitions as possible. But the initial state might take a lot of cycles to reach. Moreover, the state transitions to close the loop can act as a cool-down stage, which might reduce the average power dissipation. The second approach is to start with an initial state which is easy to reach, and return to the easy state with very few additional transitions. Hsiao et al. [8] used the second approach and pushed it to an extreme that they start from entirely *don't care* states. It takes one extra cycle to return to the initial state, since *don't care* states is superset of all states. The sequence generated by [8] would be one that has transitions from any unknown state to another state, which can be repeatedly applied to the circuit indefinitely.

14.4.2 Temperature Virus for Processor IPs

The approaches discussed in Sect. 14.3 for power virus generation for processor IPs can be directly applied to temperature virus generation. The genetic algorithm based approach can search for the synthetic program which has the best fitness with the specified system configurations. We can simply change the fitness function from peak power to sustainable power or peak temperature. The new fitness function will direct the genetic algorithm towards the search for temperature viruses.

14.5 Conclusion

In this chapter, we discussed power/temperature viruses that might cause excessive heating of IPs. At the gate level, we illustrated test generation techniques to generate automatic power viruses for both combinational and sequential circuits. At the system level, genetic algorithm can be used to automatically and effectively generate synthetic assembly programs for a given processor architecture. The test vectors and synthetic programs of power viruses can be very useful in maximum power estimation and thermal threshold selection. The power viruses can be used to test the power/thermal behaviors of a chip and the robustness of the power/thermal management design. In general, modern processor IPs have very advanced dynamic power management (DPM) and dynamic thermal management (DTM) features, such as dynamic voltage/frequency scaling [20, 21] and dynamic reconfiguration [6, 11, 18, 19, 23–25]. A power virus needs to bypass all those DPM/DTM techniques to cause real harm to the system.

References

1. R. Bertran, A. Buyuktosunoglu, M.S. Gupta, M. González, P. Bose, Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks, in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, December (2012), pp. 199–211
2. CPUburn-in. http://cpuburnin.com/
3. N.E. Evmorfopoulos , G.I. Stamoulis, J.N. Avaritsiotis. A Monte Carlo approach for maximum power estimation based on extreme value theory. IEEE Trans. Comput. Aided Des. **21**(4), 415–432 (2002).
4. K. Ganesan, L.K. John, MAximum Multicore POwer (MAMPO): an automatic multithreaded synthetic power virus generation framework for multicore systems, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage ACM International Conference for High Performance Computing, Networking, Storage and Analysis*, Article No. 53 (2011)
5. K. Ganesan, J. Jo, W.L. Bircher, D. Kaseridis, Z. Yu, L.K. John, System-level Max Power (SYMPO) - a systematic approach for escalating system-level power consumption using synthetic benchmarks. in *The 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010
6. H. Hajimiri, P. Mishra, S. Bhunia, Dynamic cache tuning for efficient memory based computing in multicore architectures, in *International Conference on VLSI Design*, 2013
7. H. Hajimiri, K. Rahmani, P. Mishra, Efficient peak power estimation using probabilistic cost-benefit analysis, in *International Conference on VLSI Design*, Bengaluru, January 3–7, 2015
8. M.S. Hsiao, E.M. Rudnick, J.H. Patel, K2: an estimator for peak sustainable power of VLSI circuits, in *IEEE International Symposium on Low Power Electronics and Design*, 1997

9. M.S. Hsiao, E.M. Rudnick, J.H. Patel, Effects of delay models on peak power estimation of VLSI sequential circuits, in *Proceedings of the 1997 IEEE/ACM International Conference on Computer-aided Design*, 1997
10. M.S. Hsiao, E.M. Rudnick, J.H. Patel, Peak power estimation of VLSI circuits: new peak power measures. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **8**(4), 435–439 (2000)
11. Y. Huang, P. Mishra, Reliability and energy-aware cache reconfiguration for embedded systems, in *IEEE International Symposium on Quality Electronic Design*, 2016
12. A.M. Joshi, L. Eeckhout, L.K. John, C. Isen, Automated microprocessor stressmark generation, in *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)* (2008), pp. 229–239 (2008)
13. Y. Kim, L.K. John, S. Pant, S. Manne, M. Schulte, W.L. Bircher, M.S.S. Govindan, AUDIT: stress testing the automatic way, in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* December (2012), pp. 212–223
14. H. Mangassarian, A. Veneris, F. Najm, Maximum circuit activity estimation using pseudo-Boolean satisfiability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **31**(2), 271–284 (2012)
15. MPrime, wikipedia page. https://en.wikipedia.org/wiki/Prime95
16. K. Najeeb, V.V.R. Konda, S.K.S. Hari, V. Kamakoti, V.M. Vedula, Power virus generation using behavioral models of circuits, in *25th IEEE VLSI Test Symposium (VTS'07)*, Berkeley, CA (2007), pp. 35–42
17. K. Najeeb, K. Gururaj, V. Kamakoti, V. Vedula, Controllability-driven power virus generation for digital circuits, in *IEEE 20th International Conference on VLSI Design (VLSID)* (2007), pp. 407–412
18. X. Qin, W. Wang, P. Mishra, TCEC: temperature- and energy-constrained scheduling in real-time multitasking systems. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. (TCAD) **31**(8), 1159–1168 (2012)
19. K. Rahmani, P. Mishra, S. Bhunia, Memory-based computing for performance and energy improvement in multicore architectures, in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2012
20. W. Wang, P. Mishra, Pre-DVS: preemptive dynamic voltage scaling for real-time systems with approximation scheme, in *ACM/IEEE Design Automation Conference (DAC)* (2010), pp. 705–710
21. W. Wang, P. Mishra, System-wide leakage-aware energy minimization using dynamic voltage scaling and cache reconfiguration in multitasking systems. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (TVLSI) **20**(5), 902–910 (2012)
22. C.Y. Wang, K. Roy, Maximum power estimation for CMOS circuits using deterministic and statistical approaches. IEEE Trans. VLSI **6**(1), 134–140 (1998)
23. W. Wang, P. Mishra, S. Ranka, Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems, in *ACM/IEEE Design Automation Conference (DAC)* (2011), pp. 948–953
24. W. Wang, P. Mishra, S. Ranka, *Dynamic Reconfiguration in Real-Time Systems - Energy, Performance, Reliability and Thermal Perspectives* (Springer, New York, 2013). ISBN: 978-1-4614-0277-0
25. W. Wang, P. Mishra, A. Ross, Dynamic cache reconfiguration for soft real-time systems. ACM Trans. Embed. Comput. Syst. (TECS) **11**(2), article 28, 31 pp. (2012)