Power and Energy Efficient Error Detection Techniques

BY

YU LIU
B.S., Dalian University of Technology, China, 2007
M.S., University of Illinois at Chicago, USA, 2011

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2013

Chicago, Illinois

Defense Committee:

Kaijie Wu, Chair and Advisor
Ashfaq Khokhar
Zhichun Zhu
John Lillis, Computer Science
Masud Chowdhury, University of Missouri, Kansas City

TABLE OF CONTENTS

SUJMMARY

Over the past decades significant technological progress has been made in Very Deep Sub-Micron and nanometer technology domains. However, the performance improvement due to shrinking size of transistors has come at the cost of decreased reliability. Our research mainly studies power and energy efficient error detection techniques at circuit and system levels.

At circuit level, we are studying the problem of synthesizing fault-secure and power efficient data path circuits from behavioral specifications. First, we propose an Integer linear programming (ILP) formulation to unify power consumption and fault security. Considering the new challenges of nanometer devices that include process parameters variations due to manufacturing imperfection and voltage variations due to manufacturing imperfection and runtime activity, we propose an ILP based comprehensive approach that considers power consumption and fault security simultaneously. The technique can adapt to the varying transient fault durations at different locations at runtime. Another challenge is the variation of fault rate of the IC chips from a same design, and time- or workload-dependent fault rate of an IC chip. To address this issue, we propose an adaptive CED technique for linear digital systems. By exploiting the linearity, the proposed CED technique performs one re-computation for *r* normal computations, where *r* is referred to as check ratio. The check ratio *r* can be adjusted on-line to perform more or less frequent CED operations according to the actual need of CED.

My research also involves studying the security concern of the scan-based Design-for-Test (DFT). Although scan-based DFT is a powerful testing scheme, we show that it can be used to retrieve the information stored in a crypto chip thus compromising its theoretically proven security.

At system level, we study power efficient task scheduling algorithm under the reliability constraint for real-time systems. The popular deployed Dynamic Voltage and Frequency Scaling (DVFS)-enabled processors can run at lower speeds to conserve energy at the cost of extended circuit delay and increased fault susceptibility. For a hard real-time system subject to faults, finding the optimal schedule of tasks that guarantees feasibility and reliability and minimizes energy consumption is of paramount significance. In

our research, we study the trade-off between reliability and energy efficiency, and propose a three-phase approach that produces a near-optimal schedule in polynomial time, $O(L^2 K \lg(K)+K^2)$ for K tasks and L voltage/frequency levels.

Chapter 1. Motivation

Over the past decades significant technological progress has been made in Very Deep Sub-Micron (VDSM) and nanometer technology domains. However, the performance improvement due to shrinking size of transistors that enables denser and smaller chips running at faster clock speeds and consuming less power has come at the cost of decreased reliability, as warned by the International Technology Roadmap of Semiconductors (ITRS). Besides inherent design defects such as ground bounce, IR drop, leakage, and charge sharing, densely packed chips are also highly susceptible to environment-induced Single Event Upsets (SEU) and Single Event Latchups (SEL). Faults occur when ions or electro-magnetic radiation strikes a sensitive device node and causes a transient or persistent change of the state of the node. To compensate for the inevitable increase of failures, and to avoid revenue losses, yield reduction, and time-to-market slowdown, the ITRS urges "automatic insertion of robustness into the design."

On the other hand, design for power efficiency is now a domain under intensive research due to increased chip density and clock frequency. Energy efficiency is crucial to many real-time systems due to their limited energy supply and severe thermal constraints of the operating environment. Power reduction techniques can help reduce power dissipation, extend battery lifetime, improve noise margin, and reduce packaging and cooling cost.

Until recently little research has been done to jointly consider both fault security and power efficiency. For systems like surveillance, satellites, and life-support implanted devices that require both fault tolerance and energy efficiency, there is a lack of efficient solutions. Simply applying fault tolerance techniques and energy minimization techniques one after the other only results in inferior quality. This is mainly due to the direct conflict between the two objectives, as fault detection is usually achieved through redundancy (space, time, or information), and a redundant system unavoidably consumes more power than its non-redundant version. This conflict must be addressed properly since the need for power efficiency and fault security continues rising while the demand for faster and better systems never stops.

However, since the techniques for the two design objectives are developed independently, they inevitably achieve one but fail the other. For example, exploiting Register-Transfer (**RT**) level operation-to-cycle schedule and operation-to-unit binding to reduce switching activities will tend to disturb the schedule and binding tuned for fault detection, and vice versa. Further, the existing circuit level power-reduction techniques do not exploit the unique characteristics of faults and redundant computations, and will not result in good quality. Similarly when we are performing frequency assignment at system level, if one minimizes energy first, i.e. allocating slack to slowing down normal task executions, the remaining slack may not be enough for fault recovery. If reservation of recovery is done before energy minimization, re-executions are treated as normal tasks and receive slacks proportionally – those slacks are wasted when faults are absent – a more possible situation than the situation where faults have occurred, according to the current fault rates of hardware systems. In fact, the two objectives cannot be achieved one after the other at both circuit level and system level. This calls for comprehensive approaches that achieve both objectives at the same time, which is the topic of our research.

Chapter 2. Circuit level techniques

2.1. Introduction

In this section, we introduce some basic concepts and assumptions used in our research.

2.1.1. High level synthesis

High level synthesis is the process of transforming a behavioral description of the system into a register transfer level (RTL) implementation. High level synthesis can be divided into several tasks, such as scheduling, binding, register allocation, interconnection determination and clock selection. We assume that the behavioral specification, which is usually provided in a hardware description language, has been compiled into a control-data flow graph (CDFG). Vertices in the graph represent operations and edges in the graph denote data dependencies. The process of scheduling assigns a start execution time to each operation in the CDFG while binding assigns a hardware unit to each operation. Clock selection refers to the procedure of choosing a value for the system clock period. An example is shown in Figure 1.

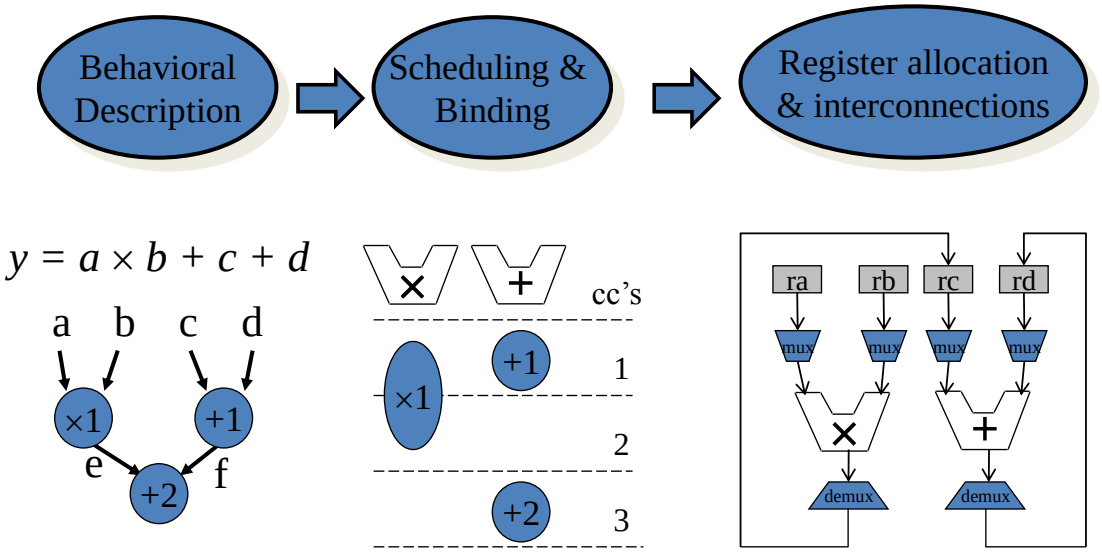


Figure 1 High Level Synthesis Flow

2.1.2. Fault security

A circuit is said to be fault-secure with respect to a specified class of faults if, on the occurrence of any fault from the class, the circuit produces either the correct output or signals an error [1][2]. The fault model that we use in this work assumes the fault to be confined to a single unit in the circuit (e.g., adder, multiplier, etc.). The fault can last for any duration (i.e., it may be permanent or transient) and can cause an arbitrary error at the unit's output [3]. Algorithm level duplication with comparison is the traditional method of providing fault security to a system [4]. The two circuits that execute the same computation will be referred to as the normal computation (NC) and the recomputation (RC). This method guarantees fault security against any fault that affects either NC or RC, but not both.

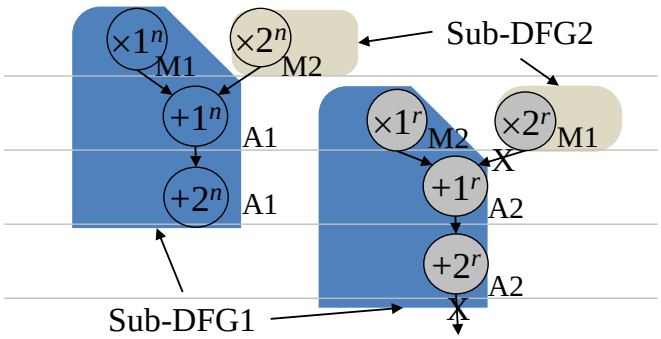


Figure 2 Hybrid time and space redundancy

Algorithm level recomputing can be implemented by time, space, or hybrid redundancy. In time redundancy based recomputing, RC has to start after the transient fault in NC dies out to satisfy fault security. It may involve significant time overhead. In space redundancy based recomputing, operations in NC and RC cannot be assigned to the same functional unit in order to satisfy fault security. Straight forward implementation of space redundancy leads to datapath duplication. Due to the disadvantages of pure time or space based techniques, hybrid time & space redundancy techniques are proposed. In these techniques, checkpoints are inserted to partition the DFG into sub-dfgs. As a result, operations in NC and RC can share resource without violating fault security if they are in different sub DFGs as shown in Figure 2. To achieve fault security, all operations must be secured. An operation is secured when the following “No-Sharing” rule is satisfied:

• *The NC copy of the datapath from the primary inputs to the output of this operation does not share any hardware units with the corresponding RC copy.*

• *The NC copy of the output of the operation is compared with the corresponding RC copy.*

In this example, checkpoint insertions create opportunities for resource minimization. In our technique, we assume the number of resources are given and insert checkpoints for power reduction. We assume that these comparison operations are performed by totally self-checking (TSC) equality checkers and, hence, faults in the units implementing these comparisons need not be explicitly considered during behavioral synthesis. Depending on how the duplicated CDFG is scheduled, many equal to comparison operations can share the same equality checker.

2.1.3. Power consumption due to switched capacitance

Power dissipation in CMOS circuits comes from two major sources: dynamic/switching power due to charging and discharging capacitive loads during state transitions, and static/leakage power due to sub-threshold currents and the reverse biased junction currents. Dynamic power due to switching activity is the concern of our research. The average dynamic power of a CMOS gate is captured as

$$P = C_L V^2 \frac{SA}{T} \propto C_L \frac{SA}{T}$$

where V is the supply voltage, C_L is the load capacitance of the gate and SA is the number of gate output transitions during the period of time T. $C_L \cdot SA/T$ represents the average switched capacitance (SC) of the gate. The total dynamic power of a circuit is

$$P_{total} = V^2 \sum C_L \frac{SA}{T} \propto \sum SC$$

A number of techniques have been proposed to reduce supply voltage, switching probability, switched capacitance, and frequency. Among these techniques, minimizing total switched capacitance by fine-tuning RT level operation-to-cycle schedule and operation-to-unit binding is the interest of our research

since it directly interacts with fault security. To illustrate how the switching capacitance is affected by scheduling and binding, consider the two schemes of schedule and binding shown in Figure 3. Suppose an application needs to perform three independent additions +1, +2, and +3 in two clock cycles, and on 2 adders A1 and A2. If the scheme of Figure 3 (a) is used, the switched capacitance equals to the switched capacitance of A1 when its input switches from +1 to +3. If the scheme of Figure 3 (b) is used, the switched capacitance equals to that of A1 when its input switches from +2 to +3. For example, if a = 0101, b = 0010, c = 0100, d = 0001, then e = a+b=0111, f = c+d=0101, g = e+f=1100. So in the scheme of Figure 3 (a) the output of A1 changes from e →g: 0111 → 1100 and 3 output ports flip while in the scheme of Figure 3 (b) the output of A1 changes from f →g: 0101 →1100 and 2 output ports flip. This shows an opportunity of optimization.

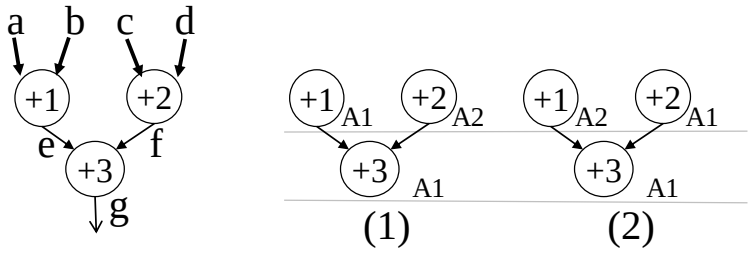


Figure 3 The effect of scheduling and binding on switching activities of a datapath

In [5], a technique to compute the switched capacitance is proposed. It uses the DFG, the input sequences and the gate level description of the function units as the inputs and generates the switched capacitance matrix. In our circuit level techniques, the objective is to minimize the total switched capacitance.

2.2. High level synthesis of fault secure datapath for power minimization

2.2.1. Motivation

Existing circuit level error detection techniques have some limitations to achieve power efficiency or fault security. Some techniques are designed to handle the worst case in order to maintain yield and runtime reliability, which leads to significant and unmanageable power overhead. For example, the n-bit RESO

adder uses extra k bits to detect the faults disturbing up to k consecutive bits [6]. It leads to a complete duplication to guarantee fault security. Other circuit level techniques are incapable to deal with the increased duration of environment-induced transient faults, i.e. SETs. Many techniques aim only at transient faults as they occur much more frequently than persistent faults. The idea is to use two or three flip-flops with shifted clocks to latch a signal [7]. If the signal is disturbed by a transient fault, the fault-caused glitch is expected to die away shortly, and will be latched by just one flip-flop – either the one with the normal clock or the one with a shifted clock, but not all. Hence comparing the states of the two (three) flip-flops detects (corrects) the fault. The idea is based on the assumption that the phase shift between clocks, which is limited to a small fraction of the clock period of the circuit, is longer than the durations of fault-caused glitches. This idea, however, is incapable for nanometer GHz designs. According to a recent study [8], environment-induced transient faults (SETs) have widths in the range of 500ps to 900ps in the 90nm process and will increase as feature size decreases. That is equivalent to several clock cycles in a multi-GHz design, which completely invalidates the assumption. Another invariance based on line test technique is proposed in [9]. The probability to miss faults is the disadvantage. The limitations of circuit level techniques motivate us to study high level synthesis of fault secure datapath for power minimization.

One of the new challenges of nanometer devices is the increased variations between wafers, dies, and the regions within a die. Those variations include process parameters variations due to manufacturing imperfection, voltage variations due to manufacturing imperfection and runtime activity, and temperature variations due to runtime activity and power dissipations. These PV variations inevitably result in various transient fault durations of the IC chips of a same design. For examples, process variations can lead to more than 10% variation of transient fault duration [10], and voltage variations can double transient fault duration [11]. For different transient fault duration, the power consumption due to switching activity of a fault secure datapath is different. Therefore, we propose a novel synthesis and design technique that

generates datapaths that meet CED constraints with minimal power consumption, and have runtime adaptability through RT level datapath control.

2.2.2. Fault duration and location aware fault security

2.2.2.1. Comprehensive scheduling and binding under fault security constraint for power optimization

Unification of scheduling and binding (S&B) can lead to designs with lower power consumption than S&B separately [6]. From Figure 2, we can see that fault security is achieved by checkpoint insertions. So next we will demonstrate the necessity to do checkpoint insertion with S&B simultaneously. Checkpoint insertion before S&B is not possible because faulty security depends on S&B. The faulty security constraint is no hardware sharing between NC and RC in the same DFG. Without the binding information, we don't have any clue to insert checkpoints. Checkpoint insertion after S&B is not good either because checkpoints allow data dependency to be broken in RC, which increases S&B flexibility and creates more opportunities to save power. In the example of Figure 4, let's assume SC is 1 between +1 and +4 and 5 for all else. After inserting a checkpoint at the output of +1, the data dependency between $+1^r$ and $+3^r$ can be broken which enables $+1^r$ to be scheduled after $+4^r$, thus saving power. Therefore, it is necessary to consider scheduling, binding, and checkpoint insertion simultaneously for power minimization.

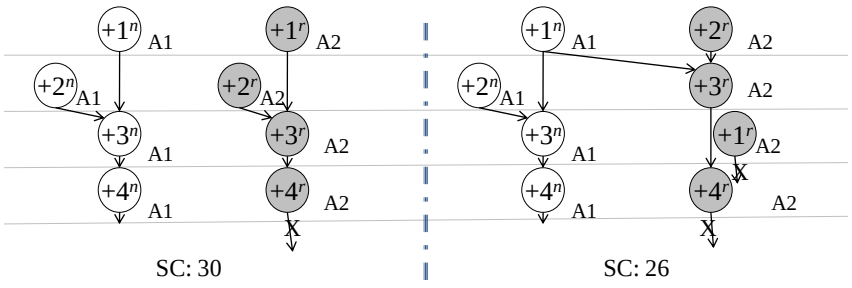


Figure 4 The effect of checkpoint insertion on power consumption

2.2.2.2. Fault duration and location aware fault security

As we have mentioned, process and voltage variations inevitably result in various transient fault durations inevitably result in various transient fault durations of the IC chips of a same design and various fault durations of the same IC over time. Recent studies have shown that the amount of multi-cycle transient faults increase with frequency and integration level of circuits. For different transient fault durations, the optimal power consumption due to switching activity of a fault secure datapath is different. Also note that the original idea of fault security is proposed for faults with any duration. So all the faults are considered as persistent, which is pessimistic. We therefore change the first part of "No Sharing" rule to that *A function unit cannot be used by both NC and RC of the same sub-DFG within N clock cycles, where N is the fault duration of the unit*. A persistent fault can be described as an N-cycle fault where N is at least two to three times larger than the total schedule length of a DFG. It is easy to see that power consumption tends to increase with N, the duration of faults. This is because with the increase of N, scheduling and binding are more constrained (even with additional checkpoints), and the resulted scheme may consume more power.

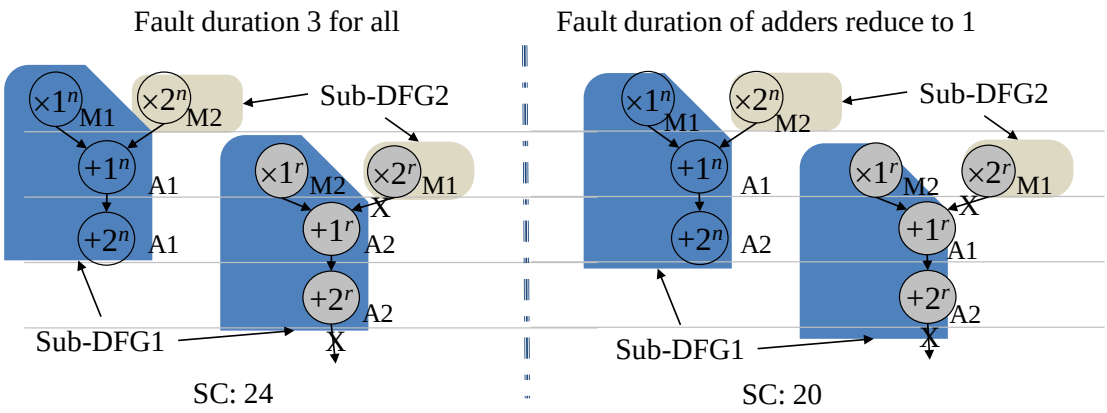


Figure 5 The effect of fault duration on power consumption

Let's look at an example as shown in Figure 5. We assume the switched capacitance is 10 and 2 respectively for multiplication and additions with different inputs. The example on the left is optimized for fault duration of 3 cycles. The fault security constraint is no sharing between NC and RC in the same sub-DFG and the total switched capacitance is 24. In the example on the right, the fault duration of adders

is reduced to 1. The transient fault is not able to corrupt two operations as long as they are scheduled 1 clock cycle apart. As a result NC and RC can be scheduled back to back for additions and the total switched capacitance 20. This example shows that fault location aware fault durations save power over a uniform worst case fault duration. It also shows that power consumption decreases with fault durations. Therefore, runtime adaptability according to different CED needs can help save power.

A reconfigurable datapath is used to execute different schemes as shown in [12]. It can adapt to a different schedule and binding in-field. A register file is used to store data, and is connected to functional units through buses. We assume the register files and buses are protected using ECC codes since they are still the most cost-effective solution. Switching between different schemes can be realized by tuning the RT level control signals: e.g., managing data to and from the buses, administrating the tri-state buffers, etc. All these can be performed at runtime. We hence focus on determining the power-optimal schedule and binding for a given CED constraint, i.e, a "No-Sharing" rule with a specific N.

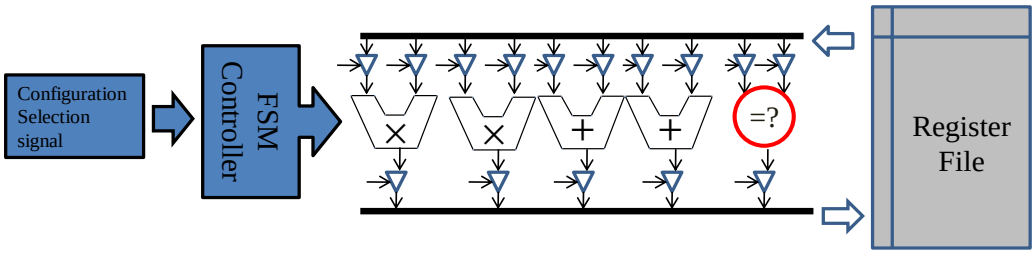


Figure 6 The effect of fault duration on power consumption

2.2.2.3. The problem formulation

Given the DFG of an application, latency and resource constraints, and a set of CED constraints derived from a set of maximal fault durations, the goal is to find, for each CED constraint, the scheme of schedule and binding that consumes the least power. In the form of general optimization problem, the problem can be described as

Objective: Minimize power consumption

Subject to: (1) scheduling and binding constraints

(2) fault security constraints.

To our best knowledge this is the first effort to unify power efficiency and fault security during high level synthesis. Since it has been shown many times that minimizing power through scheduling and binding is a NP-Hard problem in its general form [13], we will first attack this problem by formulating the problem using Integer Linear Programming (ILP), and then propose an improved heuristic approach that speeds up the process.

2.2.3. An ILP formulation to unify fault security and power efficiency[1]

Recently, a number of ILP-based approaches are proposed [14][15][16] to solve this problem and other EDA problems [17]. These approaches search for the optimal scheme of schedule and binding that minimizes power consumption subject to the constraints of resource and/or throughput. These ILP formulations have more or less similar notions. In our research, we extend them to include the constraints for fault security.

2.2.3.1. Definition

A DFG is represented by an acyclic graph $G = (O, E, d)$, where O is the set of operations, and E is the set of directed edges that represent data dependencies. O and E consist of operations and dependencies in both NC and RC. The following notations and definitions will be used in this chapter.

- L Known variable. It is the total number of clock cycles required to execute (under some constraints) all the operations in O.
- o_i, o_i^n, o_i^r o_i is the i^{th} operation in O. o_i^n and o_i^r are o_i's NC and RC copies respectively. o_i is used when we do not have to differentiate NC and RC copies.
- $s(o_i)$ Unknown variable. It is the start execution cycle of $o_i \in O$, an integer.

[1]Copyright © 2009 IEEE. This section uses the author's previous work published in 2009 IEEE International Symposium on Defect and Fault Tolerance in VLSI systems. Yu Liu, Kaijie Wu: An ILP Formulation to Unify Power Efficiency and Fault Detection at Register-Transfer Level

$d(o_i)$ Known variable. It is the delay of o_i in terms of cycles. $1 \leq d(o_i) \leq L$.

$x_{o_i,j}$ 0–1 unknown variable. It is equal to 1 if o_i starts at cycle j, or 0 if otherwise.

y_{o_i,o_j} 0–1 unknown variable. It is 1 if o_i and o_j share a same unit, and o_i executes right before o_j, or 0 if otherwise.

$SA(o_i, o_j)$ Known variable. It denotes the amount of switching power consumption of a unit by executing o_i and o_j successively.

$SA(o_i)$ Known variable. It denotes the amount of switching power consumption of a unit by executing o_i on the unit first.

c_{o_i} 0–1 unknown variable. It is 0 if there is a checkpoint at the output of operation o_i, or 1 if otherwise.

μ Known variable. It is equal to the number of different classes of units. For instance, if a design library has only adders and multipliers, μ=2.

μ_k The k^{th} unit class, where k=1, 2,…, μ. For instance, we might denote adders by μ_1, and multipliers by μ_2. $|\mu_k|$ denotes the number of the k^{th} class units.

$\tau(o_i)$ Known variable. It is the index of unit class required to execute operation o_i, i.e., o_i can only run at $\mu_{\tau(o_i)}$ units.

$z_{o_i,j}$ 0-1 unknown variable. If o_i is bound to the j^{th} unit of the $\tau(o_i)$ class, $z_{o_i,j}$= 1, otherwise $z_{o_i,j}$= 0.

P_{ck} The static power consumed by a checkpoint.

N The maximal duration of faults

In the next two sub-sections we will derive a set of constraints using these variables. Constraints introduced in section 2.2.3.2 are for general scheduling and binding. The constraints proposed in section.2.2.3.3 are dedicated for fault tolerance. The objective function is proposed in section 2.2.3.4.

2.2.3.2. The constraints for scheduling and binding

Constraint 1): An operation has a unique start execution time and should finish within the latency constraint. This constraint applies to operations in both NC and RC. Recall that $x_{o_i, j}$= 1 if and only if operation o_i starts at cycle j, we have the following constraints:

$$\sum_{j=ASAP(o_i)}^{ALAP(o_i)} x_{o_i^n, j} = 1 \qquad \sum_{j=ASAP(o_i)}^{L+1-d(o_i)} x_{o_i^r, j} = 1 \tag{1}$$

The start time of operation o_i can be derived as:

$$s(o_i^n) = \sum_{j=ASAP(o_i)}^{ALAP(o_i)} j \cdot x_{o_i^n, j} \quad s(o_i^r) = \sum_{j=ASAP(o_i)}^{L+1-d(o_i)} j \cdot x_{o_i^r, j}$$

Constraint 2) Data dependency of NC must be satisfied. This constraint only applies to the operations in NC. The constraint applicable to the operations in RC will be discussed in the next sub-section. If operation o_j^n has a dependency on operation o_i^n, we have the following constraint:

$$s(o_j^n) \geq s(o_i^n) + d(o_i) \tag{2}$$

Constraint 3) Resource constraints must be satisfied. For any operation class μ_k the user will specify the maximum number of units of this class, $|\mu_k|$, which can be used in the datapath. The maximum number of the μ_k operations that can be scheduled in any cycle t, including the operations in both NC and RC, cannot be larger than $|\mu_k|$. These operations include not only the ones that start at cycle t, but also the ones that start before cycle t and finish on or after cycle t.

$$\sum_{\{\forall o_i \in O, \tau(o_i) = \mu_k\}} \sum_{j=Max(ASAP(o_i), t-d(o_i)+1)}^{Min(ALAP(o_i), t)} x_{o_i, j} \leq |\mu_k| \tag{3}$$

Constraint 4) Unit sharing constraints. First, since an operation can only be bound to one unit, for any operation o_i, there is at most one operation that executes right before and after o_i on the same unit. Therefore for any given o_i in NC or RC, we have:

$$\sum_{\forall o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_i,o_j} \leq 1 \qquad \sum_{\forall o_i \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_j,o_i} \leq 1 \tag{4}$$

Second, y_{o_i,o_j} must be 0 if o_j starts before o_i finishes. Since the start time of operations is unknown, a variable α_{o_i,o_j} is used to constrain y_{o_i,o_j}. For any two operations o_i and o_j that $\tau(o_i) = \tau(o_j)$, α_{o_i,o_j} is defined as:

$$\alpha_{o_i,o_j} = \frac{s(o_j) - [s(o_i) + d(o_i)]}{L+1}, -1 < \alpha_{o_i,o_j} < 1$$

When o_j starts before o_i finishes, α_{o_i,o_j} will be a negative value larger than -1. Otherwise α_{o_i,o_j} will be a positive value smaller than 1. The constraint on y is as follows:

$$y_{o_i,o_j} \leq 1 + \alpha_{o_i,o_j} \tag{5}$$

Third, when the execution window of o_i and o_j overlaps (i.e. $|s(o_j)-s(o_i)| < d(o_j)$), they must be bound to different units. Therefore,

$$z_{o_i,k} + z_{o_i,k} \leq 2 + \frac{|s(o_i) - s(o_j)| - d(o_i)}{L} \tag{6}$$

When the execution window of o_i and o_j overlaps, $(|s(o_j)-s(o_i)|-d(o_j))/L$ is a negative number larger than -1. Therefore the right side of (6) is smaller than 2 and either o_i or o_j but not both can be bound to the k^{th} unit. If the execution window of o_i and o_j does not overlap, $(|s(o_j)-s(o_i)|-d(o_j))/L$ is positive and o_i and o_j may be bound to the same unit.

Next, y_{o_i,o_j} must be 0 if o_i and o_j are bound to different units,. Therefore,

$$y_{o_i,o_j} \leq 1 - \left| \sum_{m=1}^{|\mu_{\tau(o_i)}|} m \cdot (z_{o_i,m} - z_{o_j,m}) \right| \Big/ |\mu_{\tau(o_i)}| \tag{7}$$

If o_i and o_j are bound to a same unit, say the 1st unit in their class, $z_{o_i,1} = z_{o_j,1} = 1$, and $z_{o_i,m} = z_{o_j,m} = 0$ for $m \neq 1$. The right side of (7) will return 1 that allows y_{o_i,o_j} be to either 1 or 0. Otherwise, If o_i and o_j are bound to different units, say the 1st and 2nd units in their class respectively, $z_{o_i,1} = 1$, $z_{o_j,1} = 0$, $z_{o_i,2} = 0$, $z_{o_j,2} = 1$, and $z_{o_i,m} = z_{o_j,m} = 0$ for $m \neq 1, 2$. The right side will return a positive number less than 1 that forces y_{o_i,o_j} to be 0.

Each operation can be bound to only one unit.

$$\sum_{m=0}^{\left|\mu_{\tau(o_i)}\right|-1} z_{o_i,m} = 1 \tag{8}$$

All hardware units must be used.

$$\sum z_{o_i,m} \geq 1 \tag{9}$$

Finally, the ILP formulation presented in [12] suggests putting a global lower bound on unit sharing as shown below:

$$\sum_{\forall o_i, o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_i,o_j} \geq Y \tag{10}$$

The lower bound Y is either chosen by an experienced engineer or can be determined by exhaustive searching through all possible values of Y and choosing the one resulting in the lowest power [12].

2.2.3.3. The constraints for fault security

In this sub-section we will derive a set of constraints that are related to fault security.

Constraint 5) Constraints for the data dependencies in RC. If there is a checkpoint at the output of o_i (i.e. $c_{o_i} = 0$), the data dependency between o_i^r and its successor o_j^r in RC may not be satisfied as we explained in Section 2.2. Instead, o_j^r has the option to receive inputs from o_i^n. This gives a new constraint $s(o_j^r) \geq s(o_i^n) + d(o_i)$. To satisfy this constraint, we simply constrain an operation in RC starts no earlier than its corresponding copy in NC.

$$s(o_j^r) \geq s(o_j^n) \tag{11}$$

Constraint 6) Constraints for checkpoints. If the dependency from o_i^r to o_j^r is broken in a schedule, a checkpoint must be inserted at the output of o_i (i.e. c_{o_i} must be 0). Otherwise, if the dependency from o_i^r to o_j^r in RC is satisfied, there may or may not be a checkpoint (i.e. c_{o_i} may take 0 or 1). The variable β_{o_i,o_j} is used to control the dependency. β_{o_i,o_j} is defined as follows:

$$\beta_{o_i,o_j} = \frac{s(o_j^r) - [s(o_i^r) + d(o_i)]}{L}$$

It is easy to see that $-1 \leq \beta_{o_i,o_j} < 1$. β_{o_i,o_j} will be positive when the data dependency from o_i^r to o_j^r is satisfied, or negative when the dependency is broken (i.e. o_j^r starts earlier than o_i^r finishes). Therefore we have the following constraint:

$$c_{o_i} \leq \beta_{o_i,o_j} + 1 \tag{12}$$

To summarize, in a schedule if the dependency from o_i^r to o_j^r in RC is satisfied, β_{o_i,o_j} will return a positive value, which makes (12) a useless constraint since it is always true. If the dependency from o_i^r to o_j^r in RC is broken, β_{o_i,o_j} will be a negative value, which forces c_{o_i} to be 0. This means a checkpoint must be inserted.

Constraint 7) Constraints for binding that ensure fault-security. An operation in NC cannot share the same unit with its copy in RC within the fault duration of the unit N. It gives the following:

$$z_{o_i^n,m} + z_{o_i^r,m} \leq 1 + \frac{\left|s(o_i^n) - s(o_j^r)\right|}{N}, \; 1 \leq m \leq |\mu_{r(o_i)}| \tag{13}$$

The constraint further requires that any operation in a sub-DFG of NC cannot share resource with any operation in the corresponding sub-DFG of RC, and vice versa. However, the sub-DFG partitions are

unknown since the locations of checkpoints are to be determined during the searching process. We need to consider all possible pairs of operations that are of a same class with one in NC and one in RC respectively.

Recall that the variable c_{o_i} tells if o_i's output is checked. For each dependency edge from o_i, say the one from o_i to o_m, we create a new variable called $c_{o_i,\, o_m} = c_{o_i}$ since we assume there is at most one path between any two operations. In the example shown in Figure 7 there are two checkpoints. One is at the output of +1 and the other one is at the output of +3. So we have $c_{\times 1}$=1, $c_{\times 2}$=1, c_{+1} = 0, c_{+2}=1, c_{+3} = 0, and $c_{\times 1,+1}$=1, $c_{\times 2,+1}$=1, $c_{+1,+2}$ = 0, $c_{+2,+3}$=1.

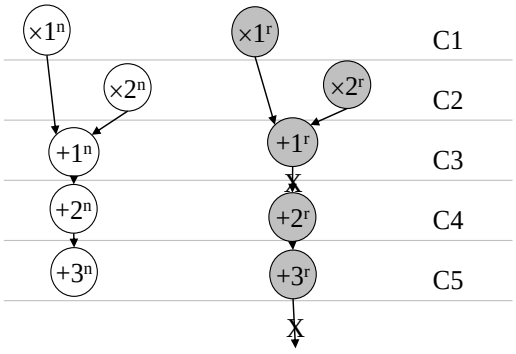


Figure 7 An example of CDFG with checkpoints

Next, to constrain the pair of operations o_i^n and o_j^r, we define the $Path_{o_i,o_j}$ as the set of edges that connect o_i^n and o_j^n by considering all the directed edges (dependencies) in NC as UN-DIRECTED edges. The constraint is only applicable to graphs with at most one path between every two nodes. In the example shown in Figure 7, $Path_{\times 1,\times 2}$ includes the dependency edge $\times 1^n \rightarrow +1^n$, and the dependency edge $\times 2^n \rightarrow +1^n$. We then have the following constraint:

$$z_{o_i^n,k} + z_{o_j^r,k} \le w + 1 - \sum_{\forall c_{o_m,o_n} \in Path_{o_i,o_j}} c_{o_m,o_n} + \frac{\left| s(o_i^n) - s(o_j^r) \right|}{N} \tag{14}$$

where w is the number of edges along $Path_{o_i,o_j}$, k ranges from 1 to $|\mu_{\tau(o_i)}|$, and N is the fault duration. If the number of checkpoints along the path is equal to or larger than 1, then the $\sum$ term will be smaller than w

and the right side of (14) will be larger than or equal to 2 which allows resource sharing between o_i^n and o_j^r. If no checkpoints along the path, then the $\sum$ term equals w and the right side of (14) will be larger than or equal to 2 only if o_i^n and o_j^r are schedule N cycles apart. Otherwise, the right side will return a value less than 2 thereby prohibiting the sharing between them. Again use $\times 1^n$ and $\times 2^r$ as an example and assume the fault duration is 2 clock cycles. The number of edges w in $Path_{\times 1,\times 2}$ is 2, and the sum of c variables along the path is 2, which indicates that the number of checkpoints in the path is 0. Thus the right side of (14) is 2+1-2+1/2=1.5 and this prohibits the sharing between $\times 1^n$ and $\times 2^r$. So, at most one of them can be bound to the k^{th} unit in the class of multiplier.

2.2.3.4. The objective function

The objective function of the ILP formulation is to minimize the sum of the switching power of function units. They are

Switching Power between operations:

$$\sum_{\forall o_i o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} SC(o_i, o_j) \cdot y_{o_i, o_j} \tag{15}$$

Cold start switching power:

$$\sum_{\forall o_i \in O} SC(o_i) \cdot (1 - \sum_{\forall o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_j, o_i}) \tag{16}$$

Power of checkpoints:

$$\sum_{\text{For all } o_i \text{ in NC}} P_{ck} \times (1 - c_{o_i}) \tag{17}$$

2.2.3.5. Experimental results

A C++ program is created to generate the equalities and inequalities for those constraints given in the above sections. The C++ program has three inputs: 1) the CDFG $G = (O, E, d)$ that gives the algorithm to be implemented; 2) the latency constraint L; and 3) the resource constraints. The ILP formulation is input to ILOG Cplex. As a comparison, we also implement the same benchmarks by applying power-reduction technique [12] first then applying fault-tolerance technique [18]. We denote such approach as Power-Before-Fault (**PBF**), and the proposed unified approach as Power-and-Fault (**P&F**). To make the comparisons fair, we use the same number of hardware resources and the same value of L for both approaches. The reason that we do not compare the approach that applies power-reduction techniques after applying fault-tolerance techniques is that the resulting datapath, with a very high probability, won't be fault secure anymore. The power dissipation of the two approaches is reported in Figure 8.

In the experiments we assume uniform delay for adders and multipliers. The power number is calculated based on the actual switching activities captured at gate level. To do so, we model the multipliers and adders using VHDL, synthesize them into gate level netlists using Synopsys DesignVision and TSMC 65nm CMOS library, simulate the netlists with random inputs using ModelSim, and record the actual gate-level switching activities. According to the report, the 16×16 multiplier chosen by Synopsys DesignVision has a range of switching activities between 0 and 510, while the 32×32 adder has a range between 0 and 120. The switching activities are then converted to power (in mw) by consulting the average dynamic power reported by Synopsys PrimePower. The static power of components is directly taken from the reports created by Synopsys PrimePower. The left y-axis in Figure 8 is for energy consumed by processing one set of inputs using PBF and P&F designs, and the right y-axis is for the difference in percentage.

As can be seen from Figure 8, the power difference between PBF and P&F is ranged from 10% up to 40%. Such a huge range is due to the unpredictability of the PBF approach. This approach first applies the power-reduction technique to NC only. RC is then added according to the technique in [12].

The running time of solving the ILP formulation increases very fast with the number of operations. The number of operations of the chosen algorithms is in the range from 10 to 14 including both NC and RC. Runtime degrades significantly for larger graphs. Interested readers can refer to [18] for more information about the correlation between the complexity of the model and the running time.

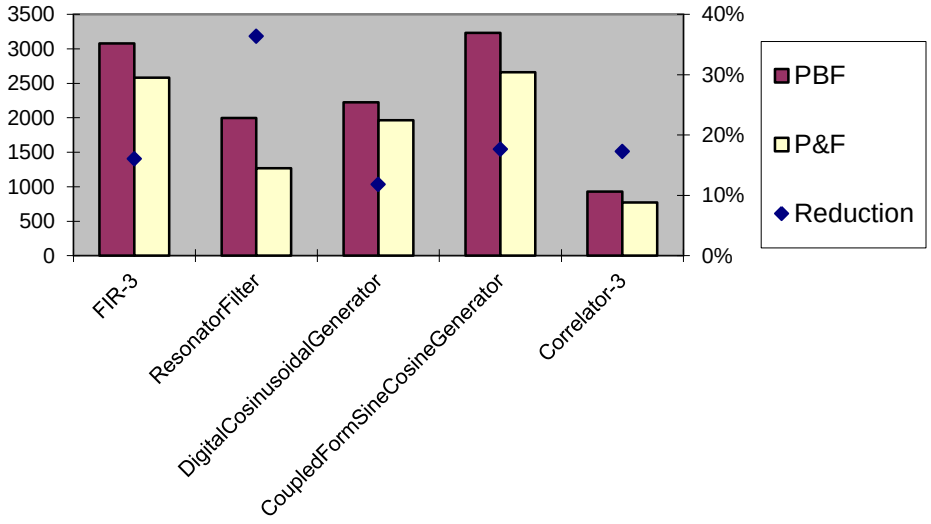


Figure 8 Power dissipation and reduction

2.2.3.6. Conclusions

In this section, we have presented an ILP formulation that minimizes power consumption of a datapath under the constraints of latency, resource, and fault tolerance. The optimal results obtained by solving the unified formulation indicate that on average 20% power reduction can be achieved over traditional non-unified techniques. Also, it can be used as references for evaluating heuristics-based approaches.

2.2.4. The improved heuristic approach[2]

2.2.4.1. Overview

[2]Copyright © 2010 IEEE. This section uses the author's previous work published in 2010 IEEE International Conference on Computer Design. Yu Liu, Kaijie Wu: Towards Cool and Reliable Digital Systems rt level CED Techniques with runtime adaptability.

Due to the nature of problem, the ILP-based solution becomes very slow when the number of operations in NC is larger than 10. A speedup is necessary for real-sized applications. Therefore, we choose to solve the problem by iterative-improvement based heuristics, which has been used with great success in the area of high level synthesis [19]-[23]. The algorithm proposed in this section is based on a general search strategy for optimization, which is called variable depth search. Examples of well-known algorithms that use this strategy are Kernighan and Lin's heuristic algorithms for graph partitioning and the traveling salesman problem [24][25].

```
1. Function: Iterative Improvement
2. Input: Initial schedule, Latency, Resource, CED constraints
3. Output: The final schedule and binding
4.  do
5.  {   Unlock all operations
6.        While there is a reschedulable operation
7.        {  Perform the move selected by Move Selection Function
8.             Compute power by the Power Estimation Function
9.             Lock the operation
10.       }
11.       Accept the sequence of moves till the one with min power
12. }until (No improvement over the previous sequence for a certain number of iterations)
```

Figure 9 The proposed iterative-improvement based heuristics

The pseudo code of the proposed algorithm is shown in Figure 9. It starts from an initial feasible schedule. This initial schedule only needs to comply with the latency and resource constraints, but not the CED constraints. Hence, it can be generated by any high-level synthesis algorithm. The algorithm looks for a sequence of schedule moves that can improve over previous iterations (Line 4-10). A schedule move is a change of an operation's start time. At the beginning of each "do" iteration, all operations are unlocked, i.e., every operation can be scheduled to another cycle if there is at least one available unit in that cycle. For a DFG with O operations and a latency of L cycles, the number of possible schedule moves are $O \times (L-1)$ in the worst case. All the possible moves are judged by a *Move Selection Function* and only the most promising move will be performed (Line 7). For the new schedule that takes the most promising move, the binding subject to resource and CED constraints is found, and the power is estimated (Line 8). This

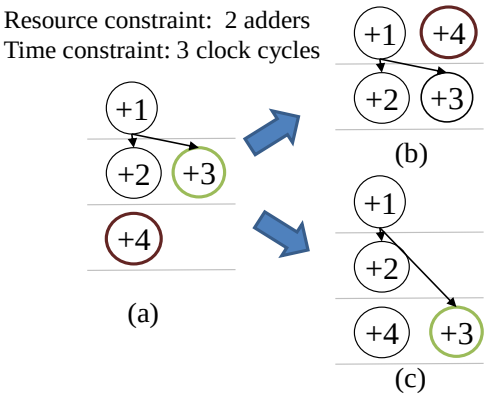
operation will be locked at the new cycle (Line 9) to avoid being moved back and forth, and the "While" loop looks for another move. Now the number of possible moves are $(O\text{-}1)\times(L\text{-}1)$ in the worst case. The "While" loop repeats until no operation is movable which generates a sequence of moves. Each move creates a new schedule by modifying the schedule created by the previous move. The schedule created by the last move is not necessarily the best one in terms of power consumption. If this happens, only the sequence of moves that reaches the best schedule will be accepted and the moves afterward will be discarded (Line 11). Since the algorithm takes the sequence of moves that has the most energy reduction in each "do' iteration even though some individual moves in the sequence may increase the energy, it is capable of hill climbing to escape from local minima. A schedulable DFG cannot contain circular dependency. So feedbacks in a DFG are broken and considered as primary inputs and outputs.

2.2.4.2. The Move Selection Function

Since the goal is to minimize power consumption, the most accurate metric is power, i.e., among as many as $O\times(L\text{-}1)$ moves, the *Move Selection Function* selects the move that results in the minimal power. However, the power of a move cannot be determined until the optimal binding of the new schedule is found. Since to find the power-optimal binding subject to resource and CED constraints is a NP-Hard problem, it is necessary to evaluate the moves in polynomial time, and then find the power-optimal binding only for the most promising move.

Our heuristic is based on the observation that *a schedule with more binding options usually leads to a binding with lower power*. A schedule with more binding variables usually has more binding options. Hence in our approach, a move is evaluated by the amount of binding variables of the new schedule, i.e., the amount of y variables in a schedule (refer to equation (7)). There are two types of switching activities, *intra-switch* and *inter-switch*. *Intra-switch* activity is generated by successively executing two operations in a same iteration instance of the DFG, and *Inter-switch* activity is generated by successively executing two operations in two successive iteration instances of the DFG. An *intra-switch y* variable is defined for

each pair of same-type operations with the following exceptions: 1) their execution windows overlap; 2) there exists at least one *Full Cycle* between the two operations. A *Full Cycle* of an operation type is the cycle where the number of operations of the type equals the number of units of the type, i.e., all units of the type are busy at a Full Cycle; 3) they are the NC and RC copies of a same operation and their schedule time is not far enough to tolerate the faults specified in the CED constraint. On the other hand, *inter-switch y* variables are for the switching activity between the operations on and after the last *Full Cycle* and the operations on and before the first Full Cycle of the schedule. If there is no *Full Cycle* in a schedule, an operation may have an *inter-iteration y* variable with itself. An example of schedule changes is shown in Figure 10. Schedules (b) and (c) are obtained by moving an operation in (a). According to our definitions, 8 y variables can be generated from (b) and 11 y variables can be generated from (c). It is not hard to see that schedule (b) has 2 binding options while schedule (c) has 4 binding options. This agrees with our statements at the beginning. The performance of the move selection approach is studied in section 2.2.4.5.



	(b)-1	*(b)-2* *(c)-1*	*(c)-2*	*(c)-3*	*(c)-4*
A1	+1, +2	+1,+3	+1,+2,+3	+1,+2,+4	+1,+4
A2	+3, +4	+2,+4	+4	+3	+2,+3

Figure 10 Schedule changes and binding options

Further, the *y* variables of different operation types may carry different weights. For example according to our experiments, the average switched capacitance between two multiplications is about 4 times that of additions. Therefore, assigning the *y* variables of multiplications a weight 4 times larger than the *y* variables of additions improves accuracy. Overall, the weighted sum of the number of *y* variables of each operation type is the metric used by the proposed *Move Selection Function*.

2.2.4.3. The Power Estimation Function

When the schedule is determined, we still need to perform binding and checkpointing for power optimization. As we have mentioned, this is still challenging because 1) When the schedule of a design is fixed, finding the power optimal binding is NP-hard [14] and 2) we will show that when scheduling and binding are fixed, inserting the minimum number of checkpoints for fault security is NP-complete. Therefore, we propose a two-step approach to attack the complexity. The first step finds the optimal binding subject to a relaxed fault security rule: a unit cannot be used for both NC and RC of the same operation within its fault duration. Note that the relaxed rule constrains an operation's NC copy and its RC copy only while the original rule constrains an operation's NC copy with all the RC operations in the same sub-DFG. The second step then inserts minimal checkpoints to the binding obtained in the first step, which enforces the original fault security rule. By inserting additional checkpoints, the binding that complies with the relaxed rule can ALWAYS be converted to comply with the original rule – any operation that violates the rule can form its own sub-DFG by inserting a checkpoint at its output. The power consumption of the scheme is the sum of the power consumption of functional units and checkpoints.

Step I. The constraints for optimal binding with relaxed fault security.

Constraint 1) Unit sharing constraints. An operation can only be bound to one unit, and one operation is carried right before and after it on the same unit. Equation (18) differs from Equation (4) in that inter-switch activities are also considered here. Hence:

$$\sum_{\forall o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_i,o_j} = 1 \sum_{\forall o_i \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} y_{o_j,o_i} = 1 \quad (18)$$

Inter-switch *y* variables add up to the number of units of that type.

$$\Sigma_{\tau(o_i)=\tau(o_j)} y_{o_j,o_i} = |\mu_{\tau(o_i)}| \quad (19)$$

Third, when the execution window of o_i and o_j overlaps, they can't be bound to a same unit as in equation (6)

y_{o_i,o_j} must be 0 if o_i and o_j are bound to different units as in equation (7).

Each operation can be bound to only one unit as in (8).

All hardware units must be used as in (9).

Constraint 2) The relaxed "No-Sharing" rule. . If the NC copy and RC copy of an operation are schedule within the fault duration of unit m, they cannot share the unit. It is obtained from equation (13) by bringing in the value of schedule time and fault duration.

$$z_{o_i^n,m} + z_{o_i^r,m} \leq 1, 1 \leq m \leq |\mu_{\tau(o_i)}| \quad (20)$$

The objective function of the ILP formulation is to minimize the sum of the switched capacitance of function units as in equation (15).

$$\sum_{\forall o_i o_j \in O, o_i \neq o_j, \tau(o_i)=\tau(o_j)} SC(o_i, o_j) \cdot y_{o_i,o_j} \quad (21)$$

The ILP formulation may have multiple binding options with the same optimal power. All the best solutions from step I are evaluated by step II. When binding for all operations are done, the minimal number of checkpoints can be determined using the ILP formulation presented below.

Step II. The constraints for optimal checkpoint insertion

Since the first step only complies with the relaxed fault security rule, additional checkpoints may be needed to enforce the original fault security rule. In order to do this, we need to check each function unit. If an operation from NC and an operation from RC in the same (sub-) DFG are bounded to the unit within

its fault duration, then at least one checkpoint must be inserted to each path between these two operations, so that the two operations are partitioned to different sub-DFGs.

After the schedule and binding are determined, the pairs of operations that violate the fault security rule on each unit can be identified and the paths between those pairs are also known. Though the complexity to compute the total number of paths between two nodes of a DAG is exponential, it is not the bottleneck of our approach. If a dependency in RC is broken, like the $\times 1^r \rightarrow +1^r$ in Figure 4, a checkpoint needs to be inserted at the broken edge which cuts the paths including the edge. Inserting the minimal number of checkpoints to cut off the remaining paths is as hard as the *Set Cover* problem – a well-known NP-Complete problem. We prove this by transforming the set cover problem to the minimum checkpoint insertion problem. Suppose we have a black box that can solve minimum checkpoint insertion. Given an arbitrary instance of Set Cover, specified by a Set U and a collection of subsets. The goal is to cover the set U with minimum subsets. We formulate an instance of minimum checkpoint insertion in which each path corresponds to an element in U. Each subset is a checkpoint. If a subset contains an element, then the checkpoint is on the path. Therefore, inserting minimal number of checkpoints that cut all the paths is equivalent to find the minimal number of these sets. The following constraint is applied to these paths:

$$\sum_{\forall \tau(o_i)=\tau(o_j), c_{o_n} \in Path_{o_i,o_j}} c_{o_n} \geq 1 \tag{22}$$

The objective is to minimize the number of checkpoints.

$$minimize \quad \sum_{\forall o_i \in O} c_{o_i} \tag{23}$$

Figure 11 is an example to illustrate how to generate the constraints from a scheduled and bound DFG. Take M1 as an example, $\times 1^n$,$\times 3^n$, and $\times 2^r$ are allocated to M1. Since they are in the same DFG within M1's fault duration, $\times 1^n$ and $\times 2^n$ needs to be partitioned to different sub-DFGs and the same for $\times 3^n$ and $\times 2^n$. Therefore, we have $c_1+c_2 \geq 1$ and $c_2+c_4+c_3 \geq 1$ respectively.

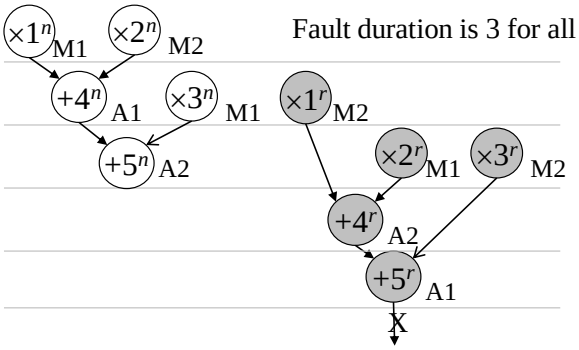


Figure 11 A scheduled and bound DFG

2.2.4.4. Checker minimization

After determining the minimum number of checking operations, we also need to determine the minimum number of checkers to execute the checking operations. We assume the comparison operations are performed by totally self-checking (TSC) equality checkers. For each checkpoint, checking can be performed from the finish time of NC and RC till the end of the iteration. Let n(t) be the number of checkpoints that start at or after t,

$$\text{Min \# of TSCs} = \max\left(n(L), n(L-1)/2, \ldots, n(1)/L\right)$$

2.2.4.5. Experimental results

We implemented the improved heuristic using C++. The program takes as inputs a latency constraint, a resource constraint, and a set of CED constraints. The initial schedule of the DFG including both NC and RC is generated before the proposed technique is applied. Since the initial schedule does not need to comply with the CED constraints, it can be produced using any scheduling and binding technique. We use the List Scheduling algorithm for the latency constrained problem. If it returns an infeasible schedule that violates the latency constraint, the program exits with a request for longer latency. During the iterative improvement, the program calls the API functions of ILOG Cplex to solve the formulations of bindings and checkpoint insertions. The switching activities of units are captured at gate level. To do so, we model the multipliers and adders using VHDL, synthesize them into gate level netlist using Synopsys

DesignVision and TSMC 65nm CMOS library, simulate the netlist with random inputs using ModelSim, and export the actual gate-level switching activities. According to the simulation results, the 16 bit multiplier chosen by Synopsys DesignVision has a range of switching activities between 0 and 510, while the 32 bit adder has a range of switching activities between 0 and 120. In our experiments, we assume the switched capacitance ranges have the same ratio. The ranges are scaled down by 10 and random numbers are generated in the range to simulate the switched capacitance. The delay of a multiplier is 2 and the delay of an adder is 1. The experiments are performed on some known benchmarks found in real-life applications: a 2nd order Differential Equation Solver (diff2) with 11 operations [27], a DTMF tone generator (dtmf) with 11 operations [28], a multiple output 2nd order filter (mof2) with 12 operations [26], a 16-point Finite Impulse Response Filter (fir16) with 33 operations, and a Correlator (cor15) with 31 operations. The numbers of operations reported above are NC only. Each benchmark takes 10 different CED constraints with $1 \leq N \leq 10$.

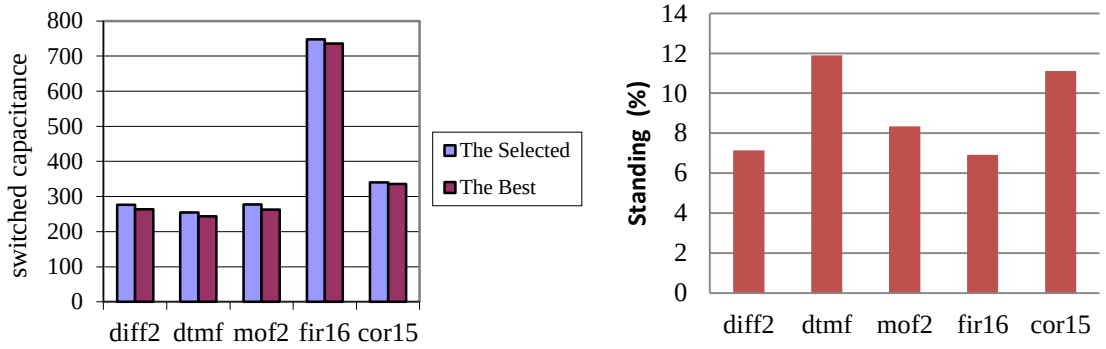


Figure 12 Accuracy of selection function

The first experiment is to verify the accuracy of the *Move Selection Function* in the proposed technique. Recall that the *Move Selection Function* bets on the move that brings the most *y* variables. To verify the accuracy of the bet, we find the optimal binding of each move of each iteration, and report the difference between the least switched capacitance among all the moves in each iteration and the switched capacitance of the selected move in that iteration. As shown in Figure 12, the top solutions are selected and the difference is less than 10% for all benchmarks. It can be observed that large differences occurred

for small benchmarks such as diff2 and dtmf. This is because that the options are limited in small benchmarks and a wrong bet could cause a larger difference.

The second experiment is to verify the effectiveness of the iterative improvement, i.e., how the iterative procedure improves over the initial schedule. The initial schedule is bound optimally and then checkpointed to comply with the same CED constraint (N=1 in this example) as the proposed technique, and the difference is reported in Figure 13. As can be seen from the figure, the proposed technique reduces switched capacitance over the initial schedule by 20% on average.

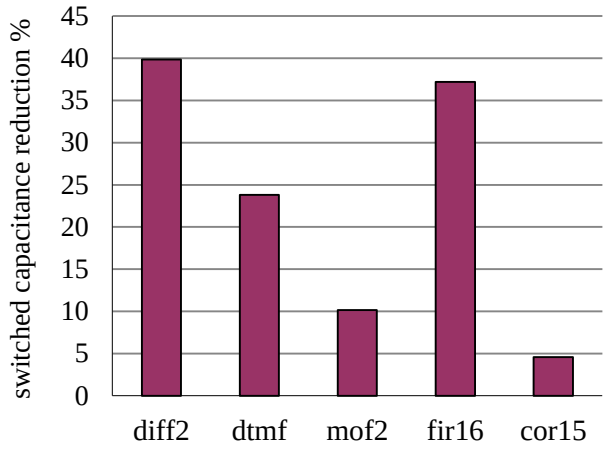


Figure 13 Improvement over the initial schedule

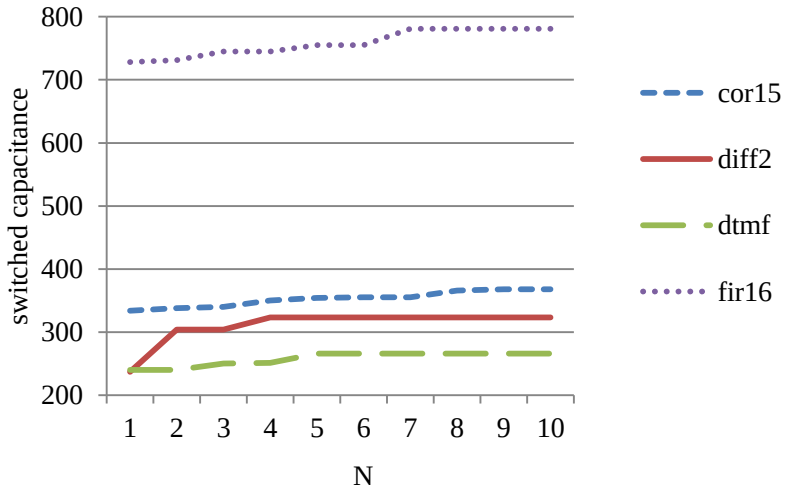


Figure 14 The power consumption over different N

The third experiment is to verify the difference of power consumption due to different CED constraints, i.e., different fault durations N. The results are reported in Figure 14. The trend shown in the figure confirms that power consumption in general increases with N especially when N is between 1 and 5. Beyond 5 all curves tend to be flat. This is because with such long durations of faults, the flexibility of scheduling and binding are strongly limited and the difference in power consumption becomes less. Comparing to the best case where N = 1, the worst case when N = 10 has 10% to 20% more switching activities.

Finally, the proposed technique can solve fairly large examples (50 operation of each type in duplicated DFG) very quickly. Comparing to the recent works that handles at most 20 operations and do not consider CED constraints (i.e., no RC copy), the runtime has been improved significantly [14]-[16]. For the DFG that has 50+ operations for a type, finding the optimal bindings by itself becomes time consuming due to its NP-Complete nature. We leave the further improvement as our future work.

2.2.4.6. Conclusions

In this section, we propose a register transfer level CED technique whose CED capabilities can be adjusted at runtime according to the actual need. The proposed high-level synthesis technique ensures that the generated datapath consumes minimal power for any CED capability it has been turned to. Experiment results show that significant amount of runtime power can be saved by our technique, and the runtime of the proposed technique is acceptable even for large examples.

2.3. Fault rate aware concurrent error detection for linear digital systems[3]

2.3.1. Introduction

Another effect of the PVT variations is on fault rate. These variations inevitably result in various fault susceptibility of the IC chips from a same design, and time- or workload- dependent fault susceptibility of an IC chip. A recent study shows that the run time variation ranges from -33.5% to 81.7% [1], and it is

[3]Copyright © 2011 IEEE. This section uses the author's previous work published in 2011 IEEE International Conference on Computer Design. Yu Liu, Kaijie Wu: Runtime adaptable concurrent error detection for linear digital systems

getting worse [31]. A SER estimation tool that models the various variations is proposed in [33]. It shows that the impact of voltage variation, temperature variation, and device aging on fault rate can be 50%. Given the fact that fault rate is low (10^{-7} to 10^{-4}) [32] and fault detection latency is always allowed, a CED technique that is adaptable to the fault rate would help save a lot of energy.

There is a thread of research that dedicates to linear digital systems. Popular examples of linear digital systems include FIR, DFT, etc. A number of information-redundancy-based CED techniques have been proposed. The basic idea is to create a Coding Matrix CM with certain properties so that errors can be detected by checking if $CM \times S(t+1) = CM \times X \times S(t) + CM \times Y \times I(t)$ [34][35][36][37][38] as shown in Figure 15. These works, however, have several rather strong limitations. First, most of the work assumes that operations never share hardware units. A 16-point FIR consisting of 17 multiplications and 16 additions thus requires 17 multipliers and 16 adders. Even though these techniques claim one check variable is enough to detect any single error in a system with n state variables, which gives just 1/n overhead, they do not consider the unrealistically high silicon cost as well as the significant static power consumption. When hardware sharing is considered, more check variables are needed which leads to more overhead. The overhead is also high if the system has a single state variable. Another strong limitation is the fault model used by these works. They assume a fault always offsets the output of the faulty unit by a fixed amount, which is not true as shown in [39].

$$\begin{pmatrix} s_1(t+1) \\ s_2(t+1) \\ . \\ . \\ . \\ s_N(t+1) \\ c\,(t+1) \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} & 0 \\ a_{21} & a_{22} & \dots & a_{2N} & 0 \\ . & . & \dots & . & . \\ . & . & \dots & . & . \\ . & . & \dots & . & . \\ a_{N1} & a_{N2} & \dots & a_{NN} & 0 \\ x_1 & x_2 & \dots & x_N & 0 \end{pmatrix} \begin{pmatrix} s_1(t) \\ s_2(t) \\ . \\ . \\ . \\ s_N(t) \\ c\,(t) \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1M} \\ b_{21} & b_{22} & \dots & b_{2M} \\ . & . & \dots & . \\ . & . & \dots & . \\ . & . & \dots & . \\ b_{N1} & b_{N2} & \dots & b_{NM} \\ y_1 & y_2 & \dots & y_M \end{pmatrix} \begin{pmatrix} i_1(t) \\ i_2(t) \\ . \\ . \\ . \\ i_M(t) \end{pmatrix}$$

Figure 15 Error detection for linear state variable system

Accumulation based CED for linear digital system is studied in [40][41][42]. Error is signaled when the accumulated offset exceeds the tolerance. In [40], fault security is not guaranteed because the bidirectional error effect can be cancelled out during accumulation. [41] and [42] study the full adder implementations for monotonic error effects. This limits the applicability of the techniques because the fast adders today are not composed of full adders.

In our research, we propose an adaptive CED technique for linear digital systems. By exploiting the linearity, the proposed CED technique performs one re-computation for r normal computations, where r is referred to as check ratio. Comparing to the existing CED techniques that are based on duplicated computations, the energy expenditure of CED is reduced significantly. By introducing coefficients and residue codes based techniques, the fault detection capability of the proposed scheme is maintained in spite of the reduced frequency of CED operations. It detects transient faults with extended durations. The check ratio r can be adjusted on-line to perform more or less frequent CED operations according to the actual need of CED. Such adaptability enables an energy-efficient CED solution for today's digital systems with strong fault susceptibility variation.

2.3.2. The basic idea

Assume $I(t)$, $O(t)$, and $S(t)$ are respectively the input, output, and state vectors of a linear system at time *t*. The transfer function of the system can be summarized as $O(t) = P \times S(t) + Q \times I(t)$, $S(t+1) = M \times S(t) + N \times I(t)$, where *M*, *N*, *P*, *Q* matrices are the arithmetic operations specified by the system. The following text focuses on $O(t) = P \times S(t) + Q \times I(t)$ only but the technique can be applied to both parts. We further simplify the expression to $O(t) = Q \times I(t)$ as our technique does not differentiate state from input. The basic idea of the proposed CED technique is on the following equation:

$$\sum_{t=1}^{r} O(t) = \sum_{t=1}^{r} Q \times I(t) = Q \times \sum_{t=1}^{r} I(t)$$

This invariance offers a great opportunity to reduce the amount of re-computations in linear systems. The idea is shown graphically in Figure 16. Let's denote a normal computation by NC and a re-computation for CED by RC. By exploiting linearity, an RC can be carried out for every r successive NCs (r=2 in this example). The r successive inputs are accumulated and fed to an RC. The result of the RC is compared with the accumulated result of the r NCs. We define r as the "check ratio." Increasing r reduces the number of RCs hence the energy consumption of CED. The detection of the faults that affect either RC or one of the r NCs is guaranteed. The implementation requires two states: NC and RC. In the state of NC, the input and output are accumulated respectively. In the state of RC, the accumulated inputs are fed to the datapath.

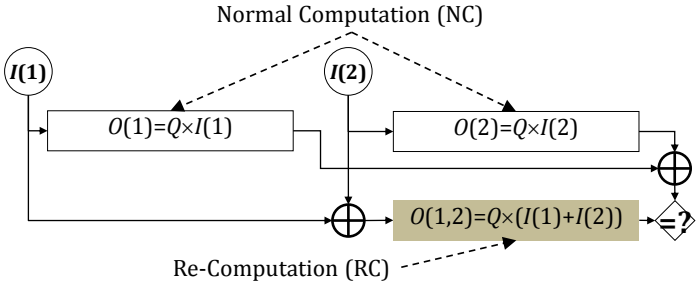


Figure 16 The basic idea

2.3.3. The run-time adaptability

As the result of the strong PVT variations in today's nanometer devices, the fault susceptibility also varies significantly [1]. Recognizing that the energy consumption of the devices with different CED capability varies significantly, designing towards the worst case of fault susceptibility (i.e., the worst-case fault rate) is not cost effective. The great benefit coming from the check ratio is that the CED capability and its energy overhead of the datapath can be changed on the fly. A set of r can be defined according to the different needs of CED that are resulted from the variation of fault susceptibility. With the run-time adaptable CED technique, one can increase r to perform less frequent CED operations in circuits with lower fault rate for more energy saving, or decrease r to perform more frequent CED operations in the

circuits with higher fault rate for better CED capability. By doing this one will never overpay the energy bill of fault tolerance.

The control of *r* could be manually by setting a desired *r* using external pins, or automatically by sensing the change of working environment such as altitude, temperature, radiation, etc. It is assumed for each different working environment the need for CED (i.e., the value of *r*) is pre-determined by offline testing. The change to *r* can be done at any time, i.e., before system starts or during runtime. A counter is implemented to track the sequence of NC. The counter is reset to 0 after an RC and increments when a NC starts and counts up to *r*-1. Before the counter reaches *r*-1, the datapath carries out NC only and accumulates the inputs and the outputs respectively. When the counter reaches *r*-1 or a value larger than *r*-1 due to a recent (runtime) change made to *r*, the datapath will transit to the other state that carries RC. The results are checked and the counter will be reset at the end of this state.

2.3.4. The problem of the basic idea

The plain implementation detects the faults that affect one of the computations, i.e., one of the *r* NCs or the RC. The proof is trivial. Hence, the transient faults lasting no more than one cycle are guaranteed to be detected. The transient faults that last several cycles, however, may affect more than one computation and may be missed. A small example is shown in Figure 17.

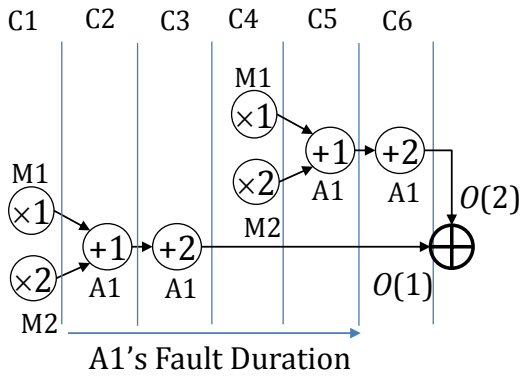


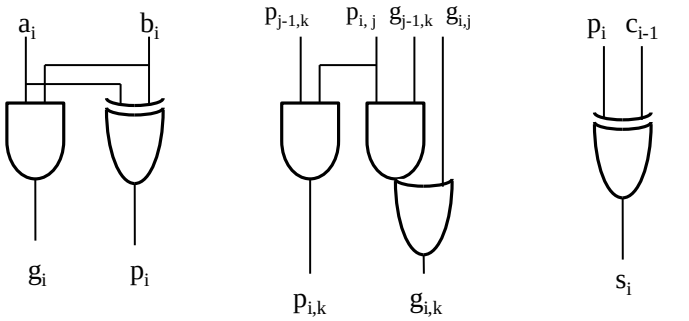
Figure 17 An example to show missed faults

The Data Flow Graph (DFG) of a small linear application has two multiplications (×1, ×2) and two additions (+1, +2). It is scheduled into three clock cycles and allocated to two multipliers (M1, M2) and one adder (A1). The DFG is repeated twice (C1-C3 and C4-C6) to represent two successive NCs. Assume the adder A1 has a transient fault that could cause one of the following offsets in the set (-2, -1, 0, +1) depending on its input [43]. The offset will be passed to the output of the computation since the computation is linear. Now we further assume the transient fault lasts from C2 to C5. Since the fault affects two operations in the first NC (i.e., +1 and +2 at cycle C2 and C3), the set of possible offsets of output $O(1)$ is (-4, -3, -2, -1, 0, +1, +2,). Since the fault affects one operation once in the second NC (i.e. +1 at cycle C5), the set of possible offsets of output $O(2)$ is still (-2, -1, 0, +1). After accumulating the (faulty) results of the two NCs, the sum of offsets could become 0 if they have the same magnitude but opposite signs. The fault will then go un-detected. There is a trivial case where the offsets at $O(1)$ and $O(2)$ are both 0, e.g. when the A1's fault occurrences at C2, C3, and C5 introduce offsets +1, -1, 0 respectively. Missing this fault does not affect system correctness. Note that a fault is said to affect an operation only if the operation's output is different from the expected output with respect to its present input, in spite that the input itself could be incorrect due to an early fault occurrence.

2.3.5. To make up the CED capability

Given a fault *f* that is identified by its location and its type (Stuck-At-0 or Stuck-At-1), the offset at the output of the host unit can be derived according to the circuit between the fault location and the output. Hence the set of offsets caused by a fault strongly depends on the implementation of the unit, and different implementations may have very different offsets. There are previous works that study the problem [43][44]. The analysis, however, only shows the magnitude of offsets without sign information. A closer investigation on the sign of offsets is performed. For example, the adders using a propagate-generate network consist of bitwise PG cells to form the generation and propagation of each bit, followed by a tree of group PG cells that form the group generation and group propagation, and then Sum cells at the last stage. The basic cells in the three stages are shown in Figure 18. It can be observed that for group

PG cells implemented using only AND gates and OR gates, a Stuck-At-0 fault can only decrease the carry out propagated through the cell while a Stuck-At-1 fault can only increase the carry out propagated through the cell. Combining this observation and the technique presented in [43] helps us determine the sign of the offsets.



(a) Bitwise PG cell (b) Group PG cell (c) Sum cell

Figure 18 The basic components in a adder using PG network

The error analysis for array multipliers and other fast multipliers are presented in [43]. These multipliers use carry save adders to compress the partial products and a carry propagate adder to generate the final product. If a full adder in the carry save adders is faulty, either sum or carry-out or both may be faulty. Then the error value is $\pm 2^k$, $1 \leq k \leq n$. If the faulty gate is in the carry propagate adder, the analysis is the same as adders.

2.3.5.1. The Coefficients Vector

To avoid cancelling the offsets during outputs accumulation when the fault affects more than one computation, we assign a set of positive coefficients to the outputs of the linear system. To balance the effect, the inputs are also multiplied by the same coefficients. The updated invariance is shown below where *CV* is the vector of coefficients.

$$\sum_{t=1}^{r} CV(t) \times O(t) = \sum_{t=1}^{r} CV(t) \times Q \times I(t) = Q \times \sum_{t=1}^{r} CV(t) \times I(t)$$

Given a fault f that is identified by its location and its type, the set of possible offsets is denoted by $S_f^1 = \{e_i^1, i \leq n^1\}$. The superscript 1 in S_f^1, e_i^1, and n^1indicates that the fault affects one operation, and n^1 is the total number of distinct offsets, i.e., the number of e_i^1. Now let's consider a simple case where the fault affects two successive NCs. If this fault affects *N* operations in the first NC, the set of possible offsets of the output of the first NC is expanded to $S_f^N = S_f^{N-1} \cup \{e_i^{N-1} + e_j^1, i \leq n^{N-1}, j \leq n^1\}$. If the fault affects *M* operations in the second NC which results in a set S_f^M, the coefficients must ensure that for any $e_i^N \neq 0$ in S_f^N and any $e_j^M \neq 0$ in S_f^M:

$$CV(1) \times e_i^N + CV(2) \times e_j^M \neq 0 \qquad \text{Eq.1}$$

Apparently if all e_i^N and e_i^M have the same sign, the inequality always holds. Hence we assume there exists at least one pair offsets (e_i^N, e_j^M) with different signs. It is easy to see that the selection of *CV* has a strong impact on the overhead. We propose to use 2^k as the coefficients so the multiplications can be implemented with very low cost. One can exhaustively search the appropriate 2^k by incrementing k starting from zero. The search process, though is exhaustive, is always upper-bounded as shown below.

To derive the upper bound, we pay attention to the maximum and minimum offsets in S_f^N and S_f^M. Let's denote the most positive offset and the most negative offset in the two sets as e_{Max+} and e_{Max-}, and the least positive offset and the least negative offset as e_{Min+} and e_{Min-} respectively. The terms "most" and "least" are according to the magnitudes of offsets. We use the following lemma to upper-bound *CV*.

Lemma 1: Eq.1 can be achieved if either of the following is true:

$$\frac{CV(2)}{CV(1)} > \text{Max}\left\{\frac{e_{Max+}}{|e_{Min-}|}, \frac{|e_{Max-}|}{e_{Min+}}\right\}$$

$$\frac{CV(1)}{CV(2)} > \text{Max}\left\{\frac{e_{Max+}}{|e_{Min-}|}, \frac{|e_{Max-}|}{e_{Min+}}\right\}$$

The proof is trivial. The idea is to magnify the least positive (or negative) offset in one set to be larger than the most negative (or positive) offset in the other set so the accumulated offset never becomes 0.

A *CV* needs to be examined on all possible faults in a unit, and on all units in the datapath. Further, faults with extended duration may affect more than two NCs. For a given datapath (i.e., the DFG is scheduled and allocated, and the implementations of units are determined), the only factor that affects the selection of coefficients is the fault durations. According to a recent study [45], environment-induced transient faults (SETs) have widths in the range of 500ps to 900ps in the 90nm process. It is about 1 to 5 cycles considering the state-of-the-art clock frequency. It is expected to grow as feature size decreases and frequency increases. The above analysis can be extended to long lasting faults that may affect more than two computations including RC.

Assume a fault spans X NCs, where it affects the q^{th} NC N_q times thereby resulting a set of offsets $S_f^{N_q} = \left\{e_i^{N_q}, i \leq n^{N_q}\right\}, 1 \leq q \leq X$. Assume NC and RC share the faulty unit [45]. The fault affects the RC N_R times thereby resulting a set of offsets $S_f^{N_R} = \left\{e_i^{N_R}, i \leq n^{N_R}\right\}$. The *CV* must ensure that for all e_i^{Nq} in all $S_f^{N_q}$, and for all $e_i^{N_R}$ in the $S_f^{N_R}$, the following inequality holds.

$$\sum_{q=1}^{X} CV(q) \times e_i^{N_q} \neq e_i^{N_R}$$

To help understand the notations, we again use the example shown in Figure 16. Assume a fault f affects the 1st NC two times and results in $S_f^{N_1} = S_f^2 = \{0,1,-1,2,-2\}$, where $q = 1$, $N_q = N_1 = 2$, and $n^{N_1} = 5$. This fault also affects the 2nd NC one time and results in $S_f^{N_2} = S_f^1 = \{0,1,-1\}$, where $q = 2$, $N_q = N_2 = 1$, and $n^{N_2} = 3$. Finally, the fault affects the RC one time and results $S_f^{N_R} = S_f^1 = \{0,1,-1\}$, where $N_R = 1$, and $n^{N_R} = 3$. To ensure the detection of the faults, one must ensure that for any $e_i^{N_q}$ in both $S_f^{N_q}$, and for any $e_i^{N_R}$ in the $S_f^{N_R}$the following inequality holds:

$$CV(1) \times e_i^{N_1} + CV(2) \times e_i^{N_2} \neq e_i^{N_R} \quad \text{Eq.2}$$

Strictly speaking, no CV works for Eq.2 if $e_i^{N_1} = e_i^{N_2} = e_i^{N_R} = 0$. This corner case can be safely ignored since the fault does not affect the computations at all. If we let $S_f^{N_0} = \{e_i^{N_0} | e_i^{N_0} = -e_i^{N_R}, i \leq n^{N_R}\}$, and $CV(0) = 1$, Eq.2 is re-written as following:

$$\sum_{q=0}^{X} CV(q) \times e_i^{Nq} \neq 0 \quad \text{Eq.3}$$

Theorem 1: Eq.3 can be achieved if the following holds:

$$\frac{CV(i)}{CV(j)} \geq \text{Max}\left\{\frac{e_{Max+}}{|e_{Min-}|}, \frac{|e_{Max-}|}{e_{Min+}}\right\} + 1, if CV(i) > CV(j)$$

Proof: Let α be the RHS. Without loss of generality we assume $\text{CV(q)} > CV(q-1)$. Hence the offset in $\text{S}_\text{f}^{\text{Nx}}$ receives the largest coefficient. Since $\text{CV}(0) = 1$, $\text{CV(q)} \geq \alpha^{\text{q}}$ for $0 \leq \text{q} \leq \text{X}$. The following deduction is to prove that after being multiplied by their corresponding coefficients, any negative (or positive) $\text{e}_\text{i}^{\text{Nx}}$ in $\text{S}_\text{f}^{\text{Nx}}$ has a larger magnitude than the sum of any positive (or negative) $\text{e}_\text{i}^{\text{Nq}}$ in any $\text{S}_\text{f}^{\text{Nq}}$ ($0 \leq \text{q} < X$), with exactly one $\text{e}_\text{i}^{\text{Nq}}$ from one $\text{S}_\text{f}^{\text{Nq}}$.

$$\frac{CV(0) \cdot |e_i^{N_0}| + CV(1) \cdot |e_i^{N_1}| + \cdots + CV(X-1) \cdot |e_i^{N_{X-1}}|}{CV(X) \cdot |e_i^{N_X}|}$$

$$\leq \frac{1 + \alpha + \cdots + \alpha^{X-1}}{\alpha^X} \cdot \text{Max}\left\{\frac{e_{Max+}}{|e_{Min-}|}, \frac{|e_{Max-}|}{e_{Min+}}\right\} = \frac{\alpha^X - 1}{(\alpha - 1)\alpha^X} \cdot (\alpha - 1) = \frac{\alpha^X - 1}{\alpha^X} < 1$$

The minimal ratio between coefficients satisfying Theorem 1 is used to upper bound the exhaustive search of *CV*. In fact, the actual ratio found by searching is much less. Section V shows more details. As can be observed, the magnitude of coefficients generally increases with the duration of faults. This translated to a wider datapath for accumulation. To reduce the overhead, we propose to use RT-level residue codes.

2.3.5.2. The residue codes

Residue code based CED circuits were initially designed for a single operator as shown in Figure 19 (a) [43][44]. The CED capability is provided by encoding operation inputs (modulation) and verifying if the output is a valid codeword. While their hardware cost is in general much cheaper than the straightforward duplication and comparison approach, the modulation and verification need to be done for each operation. In this section, we propose to check the linear datapath as a whole to reduce the overhead and improve the adaptability as shown in Figure 19 (b).

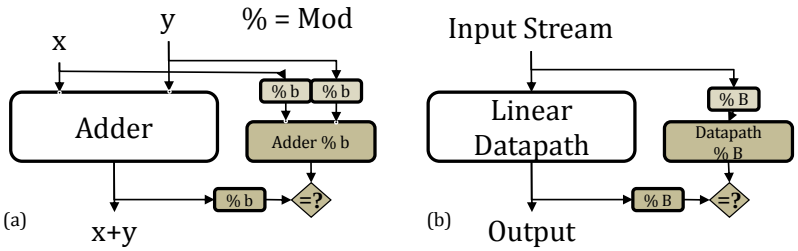


Figure 19 The residue-based CED (a) Operator oriented (b) Datapath oriented

It is easy to see that residue-based CED can be done for linear datapath since

$$O(t)\ \%\ B = Q \times I(t)\ (\%\ B) = \ (Q\ \%\ B) \times (I(t)\ \%\ B)\ (\%B)$$

The major problem in designing residue code based CED is to select the right check base B. While B should be chosen as small as possible to minimize overhead, it should detect all possible faults that could be produced by the linear datapath. If we denote the set of offsets of a fault affecting *N* operations in one computation by S_f^N, B should ensure the following for all non-zero offsets:

$$e_i^N\%\ \mathrm{B}\ \neq 0, \text{for } \forall f, \forall e_i^N \in S_f^N$$

The number of check bases that have to be examined is small since we are only interested in small check bases that have the forms of $2^k \pm 1$ for efficient residue computation. Modulo generator is an essential building block in our approach. For the check bases of type $2^k \pm 1$, modulo generator can be efficiently implemented by exploiting the properties of modulo arithmetic [47][48].

2.3.5.3. The improved idea

The block diagram of the proposed CED is shown in Figure 20. While a NC is under execution, its input is shifted, modulated, and accumulated concurrently. The accumulated inputs are then fed to RC. The output of the NC is also shifted, modulated, and accumulated. The accumulated outputs of NCs are then compared to the output of RC for error detection.

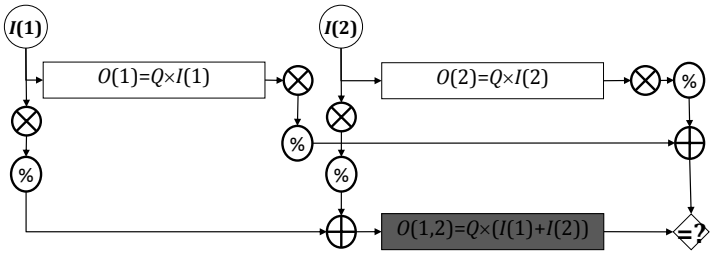


Figure 20 The block diagram of the proposed CED

Finding the appropriate *CV* and residue code is done at design (i.e. synthesis) stage. The synthesis procedure will take as inputs the worst-case fault duration and the scheduled and allocated datapath, and generate the appropriate *CV* and residue code. With the knowledge of operations' schedule and allocation, one can find *X*, the number of computations (in the worst case) that are affected by a fault with the given duration, and Nq, the number of operations in each computation that are affected by the fault. The set of offsets of each computation, S_f^N, can then be derived based on the implementation of the faulty unit. By repeating this procedure, the S_f^N of each possible fault of each unit can be derived. By searching through all the sets the e_{Min-}, e_{Min+}, and e_{Max-}, e_{Max+} and hence α can be identified. The proper *CV* is then obtained by an exhaustive search. Residue codes can then be determined. The procedure is implemented using C++. We assume all units of the same type in a datapath always use the same implementation, e.g., all adders in an implementation use ripple carry adder. Under this assumption, the search process for a 16-point Finite Impulse Response Filter takes only a few minutes.

2.3.6. Experiment results

To demonstrate the effectiveness of the proposed technique, we have implemented it on three popular benchmarks found in real-life applications: a digital cosinusoidal generator (cos) with 6 operations [29], a DTMF tone generator (dtmf) with 11 operations [29], and a 16-point Finite Impulse Response Filter (fir) with 33 operations. We use the latency-constrained List Scheduling algorithm to schedule and allocate the normal computations of these designs. Since the purpose of the experiments is to demonstrate energy saving due to reduced re-computations, we simply let the re-computations use the same schedule and allocation. The comparison is made between the proposed technique and the straightforward duplication-based CED. We model the datapaths of these benchmarks using Verilog, synthesize them into netlists using Synopsys Design Vision and TSMC 65nm device library, simulate and verify the netlists using Synopsys VCS. In our implementations, multiplications are 8-bit and take two clock cycles, and additions are 16-bit and take one cycle. We used Array Multipliers for multiplications, and Carry Ripple Adder (CRA) or Carry Lookahead Adder (CLA) for additions.

Table 1 The *CV* for the implementations using CRA/CLA

Fault Duration	3	4	5	6	7	8
cos	(2,4)	(2, 4)	(2, 8)	(4, 16)	(4, 16)	(4, 16, 64)
dtmf	n/a	(2, 4)	(2, 8)	(2, 8)	4, 16)	(4, 16)
fir	n/a	(2, 4)	(2, 8)	(4, 16)	(4, 32)/(4, 64)	(4, 32)/(4, 64)

Table 1 shows the *CV* found for the implementations of the three benchmarks. If a fault lasts less than 3 cycles in the cos implementations, or 4 cycles in the dtmf or fir implementations, it will affect only one computation. Hence no coefficient is needed. If a fault lasts 8 cycles, it could affect up to three computations in the cos implementations, or up to two computations in the dtmf and fir implementations. For the example of the implementations using CRA, a *CV* with (4, 16, 64) is thus needed for cos, a *CV* with (4, 16) is needed for dtmf, and a *CV* with (4,64) is needed for fir.

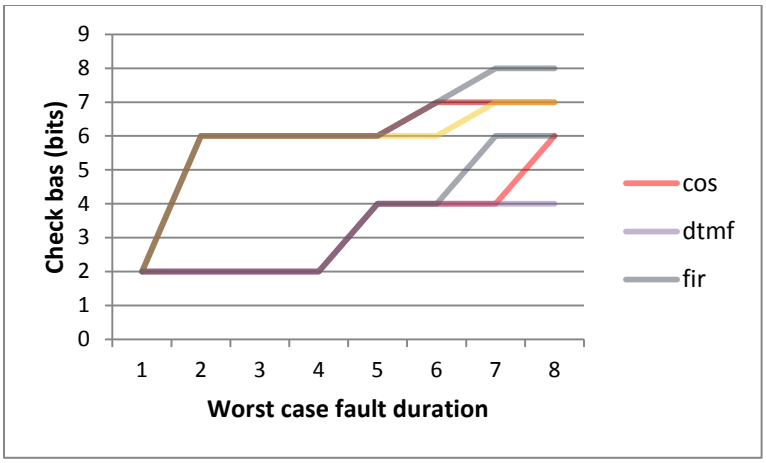
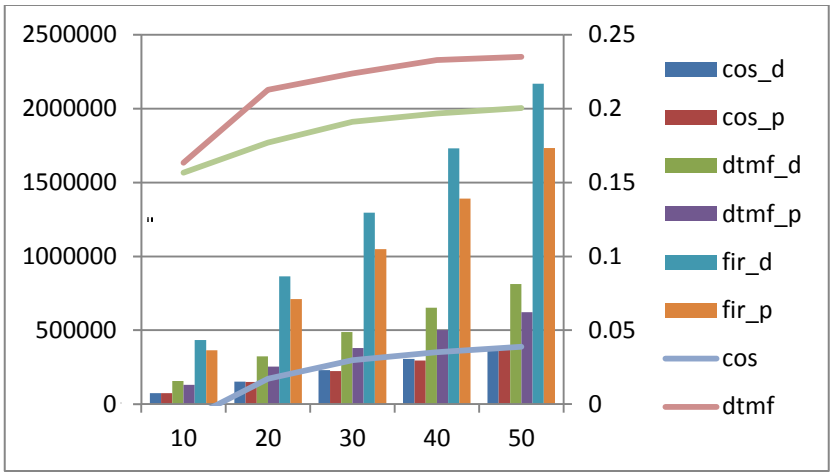


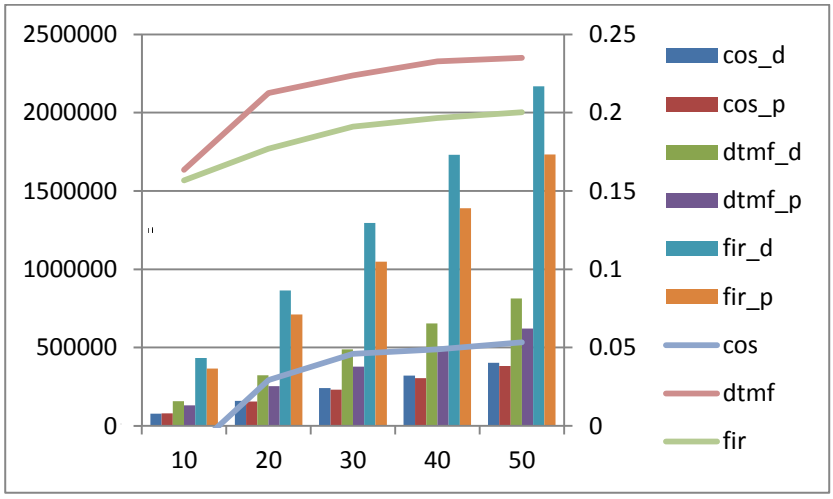
Figure 21 The width of check base increases with the worst-case fault duration

The check (modulation) bases found for the implementations are shown in Figure 21. An n-bit check base indicates that modulations are performed on either 2^n+1 or 2^n-1. As can be observed, the width of check base slowly increases with the worst-case fault duration.

Next, Figure 22 shows the energy saved by using the proposed technique. The energy saving is normalized based on the energy consumption of the duplication-based CED. For accurate power calculation, we have performed gate-level simulation and used random inputs to generate the switching activities. All implementations are designed to detect faults lasting up-to 8 clock cycles. The energy consumption are computed and the normalized energy saving is reported in the figures. From the figures, we can see that the energy saving increases with check ratio. The cos implementations have relatively small energy saving. This is majorly due to its simple DFG that consists of just 6 operations.

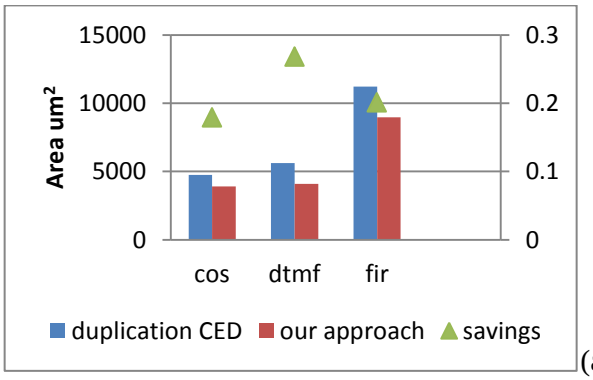


(a)

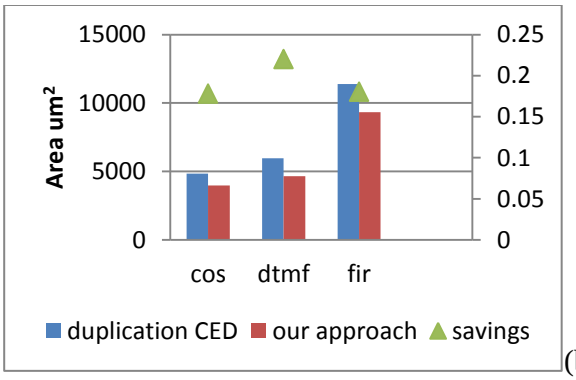


(b)

Figure 22 The normalized energy saving of the proposed technique in the implementations using (a) CRA, or (b) CLA



(a)



(b)

Figure 23 The area overhead of the proposed technique for the implementations using (a) CRA or (b) CLA

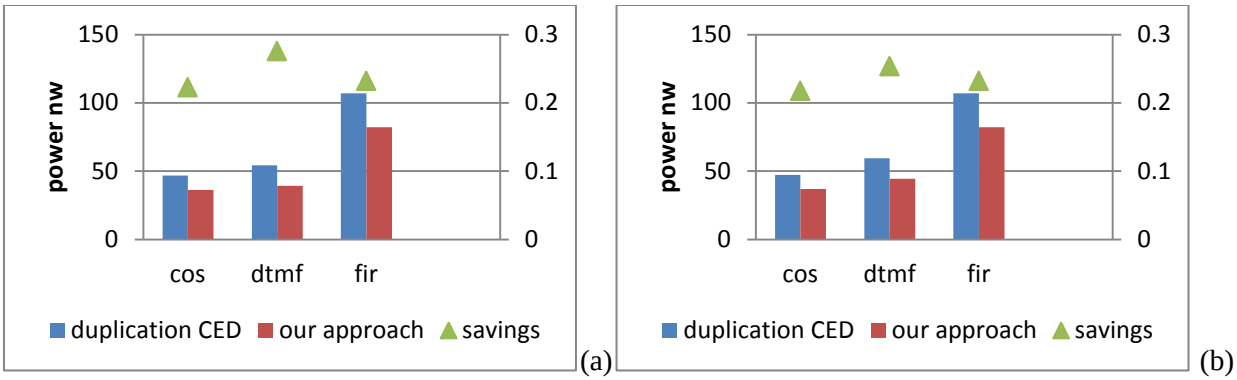


Figure 24 The static power consumption of the proposed for the implementations using (a) CRA or (b) CLA

The last comparison is on the area overhead, as shown in Figure 23. The designs are synthesized using Synopsys Design Vision and TSMC 65nm device library. The area overhead of the proposed technique includes the overhead due to the accumulations, *CV* multiplications, modulations, and the extra memory for buffering the inputs. The experimental results show that the proposed CED technique consumes much less silicon area than the duplication-based CED.

2.3.7. Conclusions

In this section, we propose a novel CED technique for linear digital systems. By exploiting the linearity, the proposed technique can handle the extended fault durations with low energy consumption. The check ratio of the proposed CED technique can be adjusted online to fit the current need of fault tolerance with minimal overhead. Experimental results show that the proposed technique saves as much as 25% energy compared with duplication CED.

Chapter 3. System level reliability analysis and fault recovery techniques

3.1. Introduction

At system level, energy efficiency is crucial to many real-time systems due to their limited energy supply and severe thermal constraints of the operating environment. Among the power- management techniques proposed to tackle these challenges, Dynamic Voltage and Frequency Scaling (DVFS) has emerged as the most popular and widely deployed scheme. DVFS dynamically adjusts the supply voltage of CMOS circuits to save power at the cost of extended circuit delays. For systems like surveillance, satellites, and life-support implanted devices that require both fault tolerance and energy efficiency, there is a lack of efficient solutions. Simply applying fault tolerance techniques and energy minimization techniques one after the other only results in inferior quality. If one minimizes energy first, i.e. allocating slack to slowing down normal task executions, the remaining slack may not be enough for fault recovery. If reservation of recovery is done before energy minimization, re-executions are treated as normal tasks and receive slacks proportionally – those slacks are wasted when faults are absent – a more possible situation than the situation where faults occurr, according to the current fault rates of hardware systems.

In our research, we propose to study the trade-off between fault tolerance and energy efficiency of fault-prone real-time systems. With a system-level reliability goal defined as the probability of finishing all tasks successfully on time in a fault-prone environment, we are interested in finding the schedule that meets the reliability goal, preserves feasibility even under the worst case of fault occurrences, and consumes the least energy when no fault occurs. A schedule in our approach includes task-to-task execution order and DVFS policy (frequency assignments of all tasks). As we will explain later that such a problem is NP-Hard, the proposed heuristic-based approach attacks this problem in three phases and produces a near-optimal schedule in polynomial time. In order to assess the quality of the solutions, an exhaustive search is implemented to find the optimal schedule since there is no other existing work on the same problem. Experimental results show that schedule produced by the proposed approach consume less than 2% more energy than the optimal solutions.

3.2. The system setup and the related works

3.2.1. The system setup

The focus of this chapter is on uni-processor real-time systems. It is assumed that the processor is DVFS capable and supports a finite set of discrete voltage/frequency levels, $F = \{f^1 = f_{max}, f^2, \ldots, f^L = f_{min}\}$. L is the number of levels and $f^l > f^{l+1}$. The system model studied in this work is referred to as frame-based systems where all tasks are released at time 0 of a time frame and should be finished before the end of the frame, as described in a survey [57]. Tasks could have dependences between them. An example of such a system is the industrial application that involves collecting data points from thousands of sensors at several MHz rate, processing them (communication, encoding or decoding, CRC check, data fusion, and etc) and then feeding the result into a decision-machine for further analysis or visualization. The read–process–analyze cycle time is determined by the rate of incoming data-points, and strict task timelines need to be followed to ensure that no data-point is missed in any cycle. Such systems have received lots of attention [53][54][60][61][62]. The deadline or duration of a frame-based system is denoted by D. For a task T_i scheduled in such systems, the following parameters are defined: N_i as its worst case CPU cycles, V_i as its operating voltage, and f_i as its operating frequency. A task set consisting of K tasks is then denoted by $TSet = \{T_i, 1 \leq i \leq K\}$.

In addition to the DVFS technique, Dynamic Power Management techniques can also conserve energy consumption by putting CPU into sleep mode when it is idle, and waking it up when tasks are pending. The idle period must be long enough to justify the additional energy and time consumed by wake-up process. It is assumed that the studied real-time systems have a short idle period after finishing the workload of the current frame. Hence the CPU will keep on (with leakage power only) until the start of the next frame. While memory is another major contributor of system energy consumption, managing memory's energy consumption requires more detailed information on the memory access patterns of tasks. Hence, this study focuses on minimizing the energy consumption of CPU only.

While environment-induced faults could be transient or persistent, it is reported that transient faults are

the most common ones – three orders higher than persistent faults [51][54][55]. Transient faults in combinational circuits are becoming as important as those in unprotected memory circuits as technology scales due to reduced voltages, nodal capacitance, clock periods, and pipeline depths [83]. This chapter focuses on transient faults, and assumes that transient faults are detected by concurrent error detection mechanisms built into the systems, such as watchdogs [79], flow signatures checking [80], simultaneously and redundantly threaded architecture [81], etc. The time overhead of fault detection is assumed to be included in the worst case execution time of tasks.

When faults are detected, the system could either roll back to the latest checkpoint or re-execute the faulty task. Since environment-induced faults are infrequent, on-demand re- executions saves considerable energy over regular checkpointing. We therefore consider re-execution-based systems. Further, the proposed technique devotes all schedule slack to slowing down normal task executions after reserving the minimal time for re-executions. This is similar to the strategy applied to checkpointing-based systems in [63]. Upon detecting faults, the system could either execute the remaining tasks on f_{max}, or reschedule the remaining tasks online according to the updated information of fault occurrences. The technique minimizes energy consumption when faults are absent and guarantees feasibility even if the worst case occurs.

3.2.2. The reliability metric

Traditionally, fault tolerance constraint, which specifies the number of faults to tolerate in a time frame, is the reliability metric used in most works on fault-tolerant systems [71][72]. In DVFS-capable systems, however, such fault tolerance constraints CANNOT be directly applied. This is because DVFS has negative impacts on the fault susceptibility of systems. As a result, running a same task set in the same environment but with different DVFS policies could have different fault susceptibilities (fault rates), hence may need to tolerate different numbers of faults to achieve the same reliability. Many recent works including [57][61][62][65][66][88] and the proposed work choose the probability-based reliability metric. It is defined as the probability of finishing a give task or task set on time in the designated working

environment [70]. It serves as a unique *Reliability Goal* (denoted by $\boldsymbol{R_G}$ thereafter) to the system that is being designed.

3.2.3. The related works and our contribution

Recently, many works on passive re-execution systems are proposed. In passive re-execution systems, re-executions are performed only when faults are detected. In 2004, Zhu et al. proposed an exponential fault rate model to capture the impact of DVFS on the rate of transient faults and showed that energy management through DVFS could impair system reliability [56]. Based on this observation they proposed to schedule an additional recovery task to recuperate the reliability loss due to DVFS for a single task model [66] and uni-core multiple tasks model [60][87]. In [58], the researchers study a similar problem based on the assumption that the re-executions are always successful. Our work explores the reliability model from another angle that does not require this simplification. The technique proposed in [62] addresses the problem in multi-core systems using Constraint Logic Programming (CLP). While the paper considers the adverse effects of DVFS on fault susceptibility, it requires the user to input both a traditional fault tolerance constraint and a probability-based reliability goal. The potential conflict is obvious since to reach the same reliability goal different numbers of faults need to be tolerated under different DVFS policies. Ejlali et al. showed that using ECC codes to protect registers in addition to re-execution can save more energy [66]. This work is different from the previous works in the following aspects:

First, being aware that the problem is NP-Hard, we propose a heuristic-based approach that can produce a near-optimal solution in polynomial time. The experimental results show that runtime is less than a second for sets with more than 1000 tasks. Hence the proposed approach can also be used to online re-schedule that collects the slack created by early completion of tasks, or to admit and schedule “emergency” tasks released in the middle of a frame. The experimental results show that the solutions from the proposed approach are close to the optimal solutions.

The second aspect is regarding the reliability metric for DVFS-capable systems. As we explained in Section 3.2.2, many recent works, including ours, use the probability-based reliability metric. There are two probability-based reliability metrics that target different levels. The task-level reliability is the probability of finishing a single *task* on time in the designated working environment, while the system-level reliability is the probability of finishing a *task set* on time in the designated working environment [66]. Many of existing works consider only task-level reliability [57][61][65][66][67][68][88]. These techniques end up reserving dedicated recovery time for each task that is not executed using f_{max}. We believe this could be too pessimistic and propose to preserve reliability at system level instead. In this work, the system-level reliability serves as a unique reliability goal (R_G) – the only reliability metric input by user. The proposed technique converts R_G to a set of traditional fault tolerance constraints where each of these constraints meets or exceeds R_G.

Finally, this work assumes that any task executed on any frequency could incur faults. This includes normal executions of tasks and re-executions of faulty tasks. The probability model proposed in [56] is used to estimate the fault rate of a task executed on a certain frequency. As a result, the achievable reliability of this work is upper- bounded only by the availability of slack. The more the slack, the higher the achievable reliability. After reserving enough slack for re-executions that achieves user-specified R_G, the rest slack can be used to slow down the processor for energy conservation. This is different from the works that try to recuperate the reliability loss due to DVFS, where the achievable reliability is upper-bounded by the probability of finishing all tasks using f_{max} without incurring faults [57][65][66][67][68][88] .

3.3. The problem and the framework

Given a uni-processor real-time system that supports a set of frequency levels ($F = \{f^1 = f_{max}, f^2, \ldots, f^L = f_{min}\}$, *and* $f^l > f^{l+1}$), *a set of K tasks, and a system-level reliability goal* R_G, *one wants to find a schedule that includes the task-to-task execution order and the DVFS policy* ($Y = (f_i \in F, 1 \le i \le K)$). *The schedule*

must satisfy the following requirements: its system-level reliability must meet or exceed R_G; it must be feasible even under the worst case of fault occurrences that is derived from R_G; it must be energy optimal when faults are absent.

The optimization problem is NP-Hard since one of its sub-problem, finding the energy-optimal DVFS policy of a task set in a system supporting discrete scaling, is in fact a Multiple Choice Knapsack Problem (MCKP). A MCKP problem is a variant of the Knapsack Problem with an additional constraint: items are subdivided into *K* classes and exact one item must be picked from each class. The problem of finding the energy-optimal DVFS policy can be easily converted to a MCKP problem: each task corresponds to a class; each class contains *L* items with each item corresponding to the task scheduled on one of the *L* frequency levels; each item has a weight and a value equal to the execution time and energy consumption of the task scheduled on that frequency, respectively. One needs to pick exact one frequency for each task (DVFS policy) with a goal to minimize the total energy consumption. Dudzinski and Walukiewicz show that MCKP is a NP-Complete problem since it is actually a special case of knapsack problem [84].

While formulating the problem into general programming approaches such as Linear or Non-Linear Programming and/or CLP is an option (for example [62]), the long and nondeterministic CPU time of these approaches is unacceptable to practical systems. A heuristic-based approach is thus highly desired.

Next, the user input reliability metric R_G needs to be converted to the form of traditional fault tolerance constraints on which the re-execution-based fault recovery schemes will be developed. However, the conversion between the two metrics is not trivial because:

The complexity of calculating the reliability of a scheduled task set: A task set is scheduled if its task-to-task order and DVFS policy are determined. For a scheduled task set comprising *K* tasks and subject to at most *X* faults, there are a total of $\sum_{x=0}^{X}\binom{K+x-1}{x}$ fault patterns [69]. The reliability can be computed by dynamic programming in polynomial time [58].

The cyclical dependence between DVFS policy and fault tolerance constraints: Since voltage scaling

negatively impacts the fault susceptibility of devices, the conversion between the two metrics becomes even more difficult. On one hand, determining a DVFS policy of a task set requires the knowledge of the amount of slack reserved for fault recovery (so that the rest can be used by DVFS to slow down). On the other hand, the slack required by fault recovery depends on the current fault rate that further depends on the DVFS policy being used.

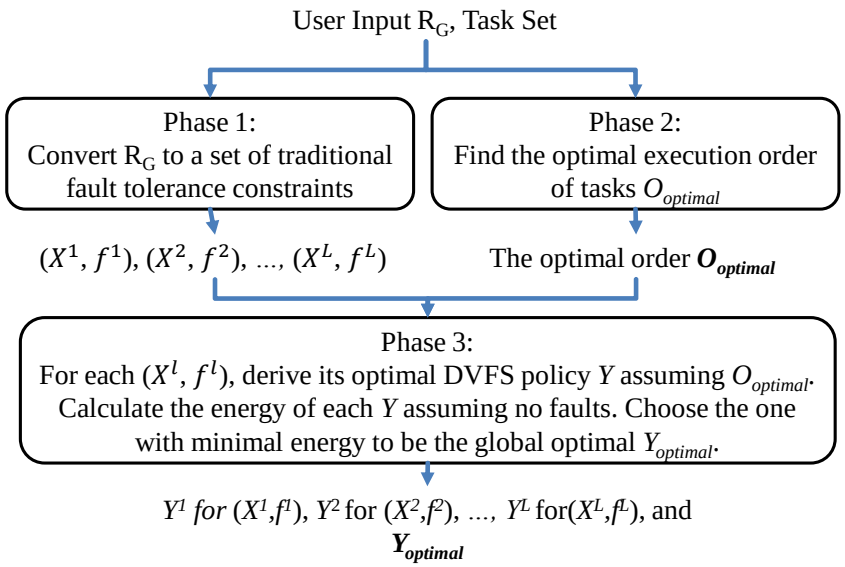


Figure 25. The proposed three-phase technique

If we denote the reliability of the system that executes tasks using DVFS policy Y and is subject to at most X faults as $R(X, Y)$, we want to test if $R(X, Y) \geq R_G$. Our approach is to calculate $R(X, Y^*)$ where policy Y^* assigns all tasks the same frequency – the lowest frequency used in policy Y. Obviously $R(X, Y) \geq R(X, Y^*)$. A tight lower bound of $R(X, Y^*)$ can be computed in constant time as shown in Section IV. The simplification, however, raises a concern that the bound could be loose if the frequencies in Y vary a lot. Thanks to the convex relation between energy and frequency, the optimal DVFS policy tends to assign tasks the same frequency or a small set of consecutive frequencies. This alleviates the problem. The proposed three-phase framework is outlined in Figure 25.

Phase 1 converts R_G to a set of traditional fault tolerance constraints: For each frequency level $f^l \in F$, we calculate the minimum X that makes $R(X, Y^*) \geq R_G$ where Y^* is a policy that assigns all tasks the same

frequency f^l. Hence the *X* guarantees the reliability for any policy *Y* that uses frequencies no lower than f^l. The calculated *X* will be recorded as a tuple (X^l, f^l). Each tuple (X^l,f^l) is a traditional fault tolerance constraint, and meets or exceeds the user specified R_G.

Phase 2 finds the optimal task-to-task execution order: This phase investigates the impact on energy due to different task-to-task execution order, and finds the order towards optimal energy saving. *Our investigation shows that the optimal order is independent to the selection of DVFS policy.* If the order of tasks cannot be changed, one can simply ignore this phase and use the designated order for Phase 3.

Phase 3 derives the DVFS policy based on the traditional fault-tolerance constraints passed from Phase 1: For each tuple (X^l, f^l) from Phase 1, Phase 3 seeks the optimal DVFS policy for the optimal order found in Phase 2. The schedule should preserve feasibility when X^l faults occur, and minimize energy when faults are absent. The optimal DVFS policy will be the one that consumes the least energy when faults are absent.

Recognizing that such a problem is NP-hard for a system supporting discrete scaling, we propose to start off with the comparable ideal system that supports continuous scaling. The analysis shows that for a given tuple (X^l, f^l) one can find the optimal policy in polynomial time in the continuous system. The optimal policy is then converted back to the discrete system using a simple heuristic.

3.4. Phase 1: System-level reliability vs. Fault tolerance constraints

This phase converts the system-level reliability goal R_G to a set of fault tolerance constraints for re-execution based fault recovery. In other words, for the system using policy *Y*, one wants to find out the minimum number of *X* so that the reliability goal is achieved, i.e. $R(X, Y) \geq R_G$. We opt for calculating the reliability of policy *Y** where all tasks are assigned the same frequency -- the lowest frequency of policy *Y*. It is easy to see that $R(X, Y^*) \leq R(X, Y)$. A tight lower bound of $R(X, Y^*)$ can be calculated in constant time [73].

3.5. Phase 2: The optimal execution order

Phase 2 is to find the optimal order of task execution. In other words, without the knowledge of this section, one may have to find the optimal DVFS policy for EVERY possible task execution order (which is *K*! in the worst case) before he can conclude which one is the best overall. With the knowledge of this section, one only needs to find the optimal DVFS policy for only one task execution order.

3.5.1. The slack constraints S_i

Feasibility is guaranteed if and only if at any time there is always enough time to handle the worst case where *X* faults are going to occur. Let us name the i^{th} running task as T_i, and denote its frequency as f_i. Consider the time instance when $T_1 \ldots T_{i-1}$ finish their executions without incurring faults, and T_i is being executed. The feasibility is ensured if, until T_i is finished, there is always enough time to handle the following situations: T_i incurs faults, or any tasks after T_i incur faults. We therefore define for each task T_i its deadline D_i and slack S_i as the following:

$$D_i = D - \left(\sum_{j=i+1}^{K} \frac{N_j}{f_{max}} + \frac{XR_i}{f_{max}} \right), \text{for } 1 \le i \le K \qquad \text{E 1}$$

$$S_i = D_i - \sum_{j=1}^{i} \frac{N_j}{f_j}, \text{for } 1 \le i \le K \qquad \text{E 2}$$

where $R_i = \max(N_i, N_{i+1} \ldots N_K)$, $1 \le i \le K$. This follows the strategy given in Section II.1 that all re-executions of the faulty task and the normal executions of all the remaining tasks are budgeted on f_{max}. S_i is determined by the assigned frequencies $f_1 \ldots f_i$. Therefore it is also denoted as $S_i(f_1, f_2, \ldots, f_i)$ when necessary. The meanings of $S_i > 0$, $S_i = 0$, $S_i < 0$ are that by executing tasks $T_1 \ldots T_i$ on the assigned frequencies $f_1 \ldots f_i$ respectively, T_i will finish earlier than its deadline, right on its deadline, or later than its deadline respectively. It is easy to see that **the sufficient and necessary condition for a policy *Y* to be feasible is for $\forall T_i \in TSet, S_i \ge 0$**. The example in Figure 26 shows the concepts of slacks and deadlines with all tasks scheduled on f_{max} and X=1. In Figure 26 a task is represented by using a bar where the

height (*y*-axis) is its frequency and the length is the execution time (*x*-axis). The deadline of the task set, *D*, is set to 15 μs. The deadline for each task, D_i, can then be calculated as D_1 = 2μs, D_2 = 6μs, D_3 = 10μs, D_4 = 12μs and D_5 = 14μs. The corresponding S_i are shown in the figure. This schedule is feasible since all tasks finish before their corresponding deadline D_i, i.e. $S_i > 0$ for $1 \le i \le 5$.

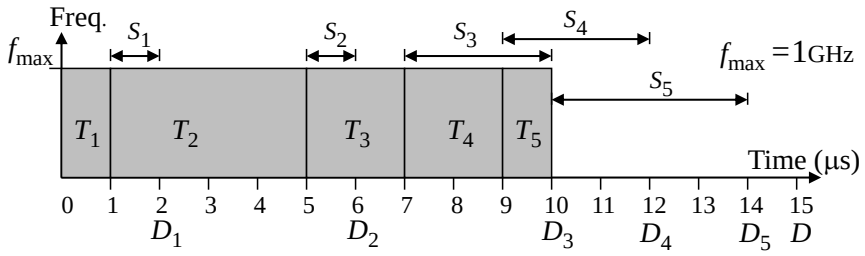


Figure 26 An example of slacks and deadlines

3.5.2. The optimal execution order

A system needs to reserve time for the worst case—the longest unfinished task incurs all the faults, and the reservation cannot be released until the longest unfinished task is executed successfully. If the longest unfinished task is executed earlier, the reservation can be released earlier and devoted to slowing down more tasks. Hence in order to minimize energy consumption, an intuition is to schedule longer tasks earlier – the order matters. The energy-optimal policies of different orders could be different, i.e., there is an optimal *Y* for each possible *O*. Among all the energy- optimal policies let us denote the one that consumes the least energy as the Global Optimal Policy and its corresponding order as the Optimal Order. The Optimal Order may not be unique.

Theorem 1: If there exists an order that always executes longer tasks earlier whenever possible (i.e., permitted by dependencies), the optimal policy of this order is always the Global Optimal Policy, and the order is one of the Optimal Orders.

Proof: Theorem 1 states that executing longer tasks earlier whenever possible is a sufficient condition. To prove it, we assume that there exists another Optimal Order *O* that does not comply with the "longer tasks

earlier" rule. We show that by converting O to a new order O^* that complies with the "longer tasks earlier" rule, the optimal policy of O, which is also the Global Optimal Policy, can still be used by the O^* without violating any slack constraints.

Assume in order O, there exists a shorter task T_i before a longer task T_j and there is no dependence between them, i.e. $O = (T_1,\ldots, T_i,\ldots, T_j,\ldots,T_K)$ and $N_i < N_j$. We move T_i after T_j to create a new order that is denoted as $O^*= (T_1\ldots T_{i-1},T_{i+1}, \ldots, T_j, T_i \ldots, T_K)$ with T_i right after T_j. We prove that the new schedule with the new order O^* and the original Global Optimal Policy Y is always feasible by showing the slack constraints of all the tasks are satisfied:

a) The slack constraints of T_m ($1\leq m\leq i-1$ or $j+1\leq m\leq K$) are satisfied as there is no change to their start and finish time;

b) The slack constraints of T_m ($i\leq m\leq j$) are also satisfied as shown below:

$$S_i^*(f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_j,f_i)$$

$$= D-\left(\sum_{k=j+1}^{K}\frac{N_k}{f_{max}}+X\frac{R_i^*}{f_{max}}\right)-\sum_{k=1}^{j}\frac{N_k}{f_k}$$

$$\geq D-\left(\sum_{k=j+1}^{K}\frac{N_k}{f_{max}}+X\frac{R_j}{f_{max}}\right)-\sum_{k=1}^{j}\frac{N_k}{f_k}$$

$$= S_j(f_1,\ldots,f_{i-1},f_i,f_{i+1},\ldots,f_j)\geq 0$$

$$S_m^*(f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_m)$$

$$= D-\left(\sum_{k=m+1}^{j}\frac{N_k}{f_{max}}+\frac{N_i}{f_{max}}+\sum_{k=j+1}^{K}\frac{N_k}{f_{max}}+X\frac{R_m^*}{f_{max}}\right)-\left(\sum_{k=1}^{i-1}\frac{N_k}{f_k}+\sum_{k=i+1}^{m}\frac{N_k}{f_k}\right)$$

$$= D-\left(\sum_{k=m+1}^{j}\frac{N_k}{f_{max}}+\sum_{k=j+1}^{K}\frac{N_k}{f_{max}}+X\frac{R_m}{f_{max}}\right)-\left(\sum_{k=1}^{i-1}\frac{N_k}{f_k}+\frac{N_i}{f_{max}}+\sum_{k=i+1}^{m}\frac{N_k}{f_k}\right)$$

$$\geq D - \left(\sum_{k=m+1}^{K} \frac{N_k}{f_{max}} + X \frac{R_m}{f_{max}} \right) - \sum_{k=1}^{m} \frac{N_k}{k}$$

$$= S_m(f_1, \ldots, f_{i-1}, f_i, f_{i+1}, \ldots, f_m) \geq 0 \ \ (i < m \leq j)$$

In this deduction, R_i^* = max (N_i, N_{j+1},…,N_K) ≤ R_j = max (N_j, N_{j+1},…,N_K), and R_m^* = max (N_m,…, N_j, N_i, N_{j+1},…,N_K) = R_m = max (N_m, …,N_j, N_{j+1},…,N_K) since $N_i < N_j$. $S_j \geq 0$ and $S_m \geq 0$ are because the original schedule is feasible.

Energy wise, since the new schedule also uses the original Global Optimal Policy *Y*, the new schedule consumes the same energy as the original schedule. Hence the new order *O** is an Optimal Order. **Q.E.D.**□

```
Function: Find_Optimal_Order (TSet)
  USet ← TSet
  while USet <> ∅ do
    Find the longest task in USet
    if the task has unscheduled ancestors then
        Find_Optimal_Order(Fan in tree of the task)
    end if
    Schedule the task, and Remove the task from USet
  end while
```

Figure 27. Find the Optimal Order

Theorem 1 greatly simplifies the searching space of execution orders from as many as *K*! to only one – the one that always schedules longer tasks earlier. This order will be referred to as the Optimal Order thereafter. The pseudo code is given in Figure 27. Initially, all tasks are unscheduled and are listed in *USet*. Each of the following **While** iteration schedules the longest task in *USet* and the tasks on which it depends. The total time complexity is $O(K^2)$.

3.6. Phase 3: The optimal DVFS policy

A tuple (X^l,f^l) passed from Phase 1 delivers two constraints: X^l is the maximum number of faults to tolerate in a time frame, and f^l is the minimal frequency that can be used in the policy. For the purpose of this section, f^l and $X^l$are referred to as f_{sch_min} and X respectively. The frequency (f) of executing a task is a function of the voltage (V) on which the processor is running: $f = b(V - V_{th})^2 / V$, where V_{th} is the threshold voltage and b is a constant for a given technology process. Mathematically, the above equation can be rewritten as if V is a function of f: $V = V(f)$, where function $V(\cdot)$ is a monotonically increasing function of f when $V > V_{th}$. Hence, the power consumption of an active processor can be determined by its frequency, as denoted by $P_A(f)$, and $P_A(f) = P_d(f) + P_{ind}$, where $P_d(f)$ and P_{ind} are frequency-dependent power and frequency-independent power respectively [60]. The energy consumed by executing a task with N_i CPU cycles at frequency f_i can then be computed as $\frac{P_A(f_i)N_i}{f_i}$.[4] The total energy consumed by executing K tasks is then given by:

$$E_{Active} = \sum_{i=1}^{K} \frac{P_A(f_i)N_i}{f_i} \qquad \text{E 3}$$

After finishing all scheduled tasks, the processor is assumed to go into idle/sleep state where only the minimal static power ($P_{Idle} \geq 0$) is consumed:

$$E_{Idle} = P_{Idle}\left(D - \sum_{i=1}^{K} \frac{N_i}{f_i}\right) \qquad \text{E 4}$$

Therefore, the total energy consumption of executing K tasks in a processor in a time frame with duration D is:

[4] We assume system voltage/frequency changes only at the start of a task and will stay until the end of the task.

$$E_{Total} = E_{Active} + E_{Idle} = \sum_{i=1}^{K} \frac{P_A(f_i)N_i}{f_i} + P_{Idle}\left(D - \sum_{i=1}^{K} \frac{N_i}{f_i}\right) = \sum_{i=1}^{K} \frac{(P_d(f_i) + P_{ind} - P_{Idle})N_i}{f_i} + P_{Idle}D \quad \text{E 5}$$

Let $E(f_i) = \frac{P_d(f_i)+P_{ind}-P_{Idle}}{f_i}$ represent the energy consumption per clock cycle, E_{Total} can be written as:

$$E_{Total} = \sum_{i=1}^{K} E(f_i)N_i + P_{Idle}D \quad \text{E 6}$$

$P_d(f)$ is believed to be a strictly increasing convex function of f, represented by a polynomial with an order between 2 and 3 [74]. Since P_{Idle} is the minimal static power consumed when the processor is idle, it can be derived that $P_{Idle} \leq P_{ind}$. Therefore $E(f)$ is a convex function and the minimum system energy is achieved when f takes the so-called *critical* frequency[62]. Since executing below the critical frequency consumes more time and energy, all the frequencies considered in this study are no less than it.

Determining the optimal DVFS policy for a system supporting discrete scaling is NP-Hard. We propose to first investigate a comparable system that supports continuous scaling. The optimal policy for the continuous system can then be derived in polynomial time, and will be converted back to the original discrete system using a simple heuristic.

3.6.1. Continuous system: Determine the optimal DVFS policy of a given execution order

3.6.1.1. The algorithm

The example in Figure 28 is used to demonstrate the meanings of these notations. Consider a task set comprising of 5 tasks that are named according to the execution order: T_1, T_2, T_3, T_4, and T_5, and require 1k, 4k, 2k, 2k, 1k CPU cycles (N_i), respectively, as shown in Figure 28 (a). The processor has f_{max} = 1GHz and can scale down to f_{min} = 300MHz continuously. The system is supposed to tolerate at most 1 fault in a

time frame. In Figure 28, a task is represented by a bar where the height (*y*-axis) is its frequency and the length (*x*-axis) is its execution time. The deadline of the task set, *D*, is set to 15 μs. The deadline of each task, D_i, can then be calculated as D_1 = 2μs, D_2 = 6μs, D_3 = 10μs, D_4 = 12μs and D_5 = 14μs. In Figure 28 (a), all tasks are executed on f_{max}, and such a policy is feasible since all tasks finish before their corresponding deadlines, i.e. $S_i > 0$ for $1 \le i \le 5$. The goal is to find the optimal DVFS policy of the task set so that the energy is minimized without letting $S_i < 0$, for $1 \le i \le 5$.

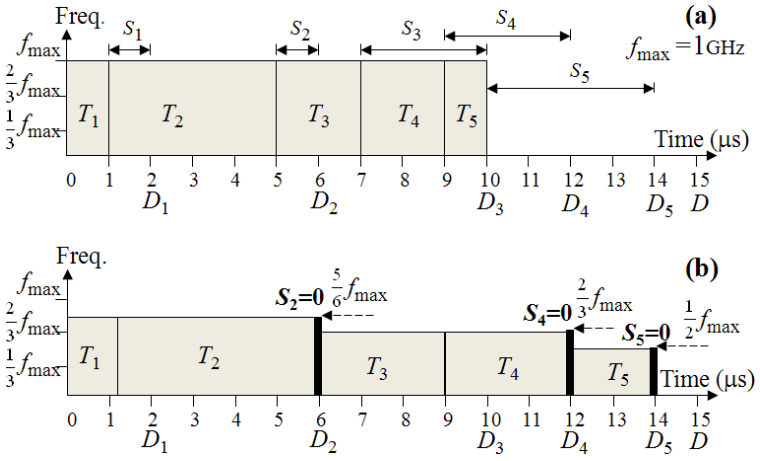


Figure 28. An example with f_{sch_min} = 1/3 f_{max} and X = 1

We propose the following iterative procedure to find the optimal DVFS policy. The proof of optimality follows in the next subsection. *I* is set to 0 at the beginning:

1) Reduce the frequencies of all tasks scheduled AFTER the I^{th} task simultaneously until either f_{min} is reached or the slack of any task decreases to 0. If it is the former case, the procedure returns with a complete policy. Otherwise, go to 2).

2) Update *I* using the index of the task that runs out slack. If more than one tasks run out slack at the same time, *I* is set to the last one. Go to 1).

In the example, T_2 is the first task that runs out slack, followed by T_4, then T_5. The complete policy is Y = {f_1=833MHz, f_2=833MHz, f_3=667MHz, f_4=667MHz, f_5= 500MHz}, as given in Figure 28 (b).

3.6.1.2. The proof of the optimality

From the previous example, it can be seen that the frequencies of tasks produced by the iterative procedure always have $f_i \geq f_{i+1}$. Theorem 2 shows that this is a necessary condition for a DVFS policy to be energy optimal.

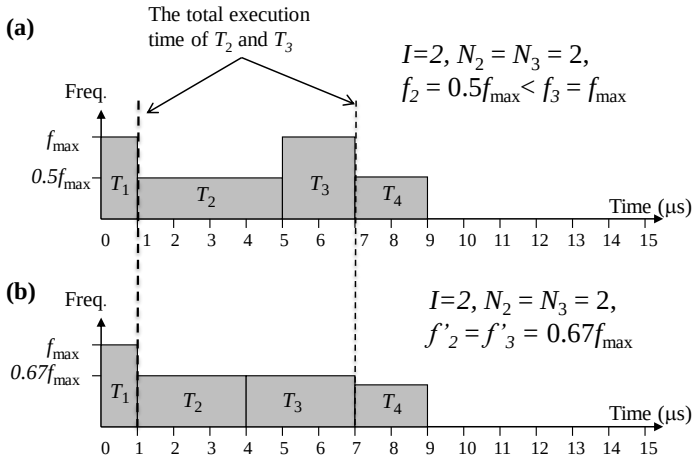


Figure 29. An example to show $f_I \geq f_{I+1}$ is necessary

Theorem 2: For $TSet = \{T_i, 1 \leq i \leq K\}$, if a policy $Y=(f_i, 1 \leq i \leq K)$ is feasible and energy-optimal, then $f_i \geq f_{i+1}$.

Proof: We make a counter statement that in an "optimal" policy there exists an index I such that $f_I < f_{I+1}$. Then we can build a new policy that assigns the following f'_I and f'_{I+1} to tasks T_I and T_{I+1} respectively:

$$f'_I = f'_{I+1} = \frac{N_I + N_{I+1}}{\frac{N_I}{f_I} + \frac{N_{I+1}}{f_{I+1}}}$$

It is trivial to prove $f'_I > f_I$, and $f'_{I+1} < f_{I+1}$. Figure 29 shows a small example for the proof and T_I and T_{I+1} are the 2nd and 3rd tasks respectively. Figure 29 (a) is the original "optimal" frequency that has $f_2 < f_3$, and Figure 29 (b) is the new policy that has $f'_2 = f'_3$. Comparing to the original "optimal" policy, the total execution time of tasks T_I and T_{I+1} of the new policy remains the same, and hence the feasibility of the tasks except T_I remains the same. Since T_I finishes earlier, its feasibility is not hurt either. So the new policy is feasible.

Again comparing to the original "optimal" policy, the energy consumed by the tasks except T_I and T_{I+1} remains the same since their frequencies are not changed. But the total energy consumed by T_I and T_{I+1} in the new policy is less. This is because $E(f)$ is a convex function and the total execution time of T_I and T_{I+1} remains the same, executing tasks T_I and T_{I+1} using f_I' and f_{I+1}' always consumes less energy than using f_I and f_{I+1} [59][86]. Therefore the new policy consumes less energy than the original one. This disproves the counter statement thereby proving this lemma. **Q.E.D**.

Task T_i is called a breakpoint (**BP**) task if $f_i > f_{i+1}$. In the example given in Figure 28 (d), T_2 and T_4 are BP tasks. Those BP tasks are of special interest to us since they define the complete policy of all tasks. We create a Δ variable for each task as in Eq 5:

$$\Delta_i = \frac{\sum_{k=1}^{i} N_k}{D_i}, \text{for } 1 \le i \le K \qquad \text{E 7}$$

It can be seen as a hypothetical frequency for task T_i in the way explained by Lemma 1.

Lemma 1: The slack constraint of T_i will be satisfied if T_i and all the tasks before T_i are executed on frequencies equal to or higher than Δ_i, i.e. $S_i \ge 0$ if $f_k \ge \Delta_i$ for $1 \le k \le i$; The slack constraint of T_i will be violated if at least one task, either T_i or any task before T_i, is executed on a frequency lower than Δ_i while none of the other tasks is executed on frequencies higher than Δ_i, i.e. $S_i < 0$ if $f_k \le \Delta_i$ for $1 \le k \le i$ and $\exists j$, $1 \le j \le i$, $f_j < \Delta_i$.

Proof: The proof is straightforward.

If $f_k > \Delta_i$ for $1 \le k \le i$

$$S_i(f_1, \dots, f_i) = D_i - \sum_{k=1}^{i} \frac{N_k}{f_k} > D_i - \sum_{k=1}^{i} \frac{N_k}{\Delta_i} = 0$$

If $f_k > \Delta_i$ for $1 \le k \le i$

$$S_i(f_1, \ldots, f_i) = D_i - \sum_{k=1}^{i} \frac{N_k}{f_k} = D_i - \sum_{k=1}^{i} \frac{N_k}{\Delta_i} = 0$$

If $f_k \le \Delta_i$ for $1 \le k \le i$, and $\exists j$, $1 \le j \le i$, $f_j < \Delta_i$

$$S_i(f_1, \ldots, f_i) = D_i - \sum_{k=1, k \ne j}^{i} \frac{N_k}{f_k} - \frac{N_j}{f_j} < D_i - \sum_{k=1}^{i} \frac{N_k}{\Delta_i} = 0$$

Q.E.D. □

Lemma 2 and Theorem 3 show that the first BP task is the one with the largest Δ value among all the tasks.

Lemma 2: For a feasible and energy-optimal policy $Y=(f_i, 1 \le i \le K)$, if task T_I is the first BP task and $\Delta_I > f_{sch\_min}$, then $f_k = \Delta_I$ for $1 \le k \le I$.

Proof: If task T_I is the first BP task, all the tasks scheduled before T_I are on the same frequency as T_I, i.e. $f_k = f_I$ for $1 \le k \le I$. Next, Lemma 1 tells that the frequency of these tasks cannot be less than Δ_I due to slack constraints, i.e. $f_k \ge \Delta_I$ for $1 \le k \le I$. Therefore we only need to show that f_k is no larger than Δ_I for $1 \le k \le I$.

We then make the counter statement that the "optimal" policy has $f_k > \Delta_I$ for $1 \le k \le I$. Especially, $f_I > \Delta_I$. If T_I is the last task, i.e. $I = K$, the policy cannot be optimal because f_I can always be reduced without affecting the other tasks. Hence we will focus on the case where T_I is not the last task, i.e. $I<K$. Since it is assumed that the processor supports continuous scaling, one can always build a new policy that assigns frequencies, f_I', f_{I+1}', to tasks T_I and T_{I+1} respectively such that

$$f_I' > f_{I+1}', \text{ and } f_I' = f_I - \varepsilon > \Delta_I, \varepsilon > 0$$

E 8

$$\frac{N_I}{f_I} + \frac{N_{I+1}}{f_{I+1}} = \frac{N_I}{f_I'} + \frac{N_{I+1}}{f_{I+1}'}$$

Comparing to the original "optimal" policy, the total execution time of tasks T_I and T_{I+1} of the new policy remains the same, and hence the feasibility of the other tasks except T_I and T_{I+1} remain the same. Also because the frequencies of T_I and all the tasks executed before T_I have the following property: $f_1 = \ldots = f_{I-1} > f_I' > \Delta_I$, according to Lemma 1, the slack constraint of task T_I is satisfied, i.e. $S_I' \geq 0$. $S_{I+1}' = S_{I+1} \geq 0$ since T_{I+1} finishes at the same time in the new policy. So the new policy is feasible.

Again comparing to the original "optimal" policy, the energy consumed by the tasks except T_I and T_{I+1} remains the same since their frequencies are not changed. But the energy consumed by T_I and T_{I+1} in the new policy are less. This is because $E(f)$ is a convex function and the total execution time of T_I and T_{I+1} remains the same, executing tasks T_I and T_{I+1} using f_I' and f_{I+1}' always consumes less energy than using f_I and f_{I+1} according to theorem 3 in [59]. Therefore the new policy consumes less energy than the original one. This disproves the counter statement thereby proving this lemma. **Q.E.D**. □

Theorem 3: For a feasible and energy-optimal $Y=(f_i, 1\leq i\leq K)$, T_I is the first BP task if $\Delta_I > f_{sch\_min}$, and $\Delta_I > \Delta_k$ for $I<k\leq K$, and $\Delta_I \geq \Delta_k$ for $1\leq k<I$.

Proof: The theorem assumes task T_I, the task that has the largest Δ frequency, is not the last task, i.e. $I<K$. The theorem is similar and the proof is easier if T_I is the last task. We make a counter statement that the first BP task is T_J, $J\neq I$, $J\leq K$.

If $J < I$, we have $f_1 = \ldots = f_J > f_{J+1} \geq \ldots \geq f_I$. According to Lemma 1, f_J has to be greater than Δ_I. Together with the condition that $\Delta_I \geq \Delta_k$ for $1\leq k<I$, it can be derived that $f_J > \Delta_J$.

If $J > I$, we have $f_1 = \ldots = f_I = \ldots = f_J > f_{J+1}$. According to Lemma 1, $f_I \geq \Delta_I$, hence $f_J = f_I \geq \Delta_I$. Together with the condition that $\Delta_I > \Delta_k$ for $I<k\leq K$, it can be derived that $f_J > \Delta_J$.

Therefore the counter statement always leads to $f_J > \Delta_J$. According to Lemma 2, T_J cannot be the first BP task. This contradicts the original statement thereby proving Theorem 3 **Q.E.D.** □

Therefore if T_I is the first BP task, letting $f_i = \Delta_I$ for $1 \le i \le I$ will lead to the feasible and energy-optimal policy. However, consider the second BP task, say T_J ($I < J$), we CANNOT simply let $f_j = \Delta_J$, for $I < j \le J$. This is because Δ_J is calculated by assigning Δ_J to all the first J tasks. But it has been proven that all the first I tasks will be executed on Δ_I, a frequency higher than Δ_J. Therefore to determine a new BP task, one may consider a new frame that starts after the most recent BP task finishes, and a new task set that consists of all tasks after it. The Δ frequency of these tasks will be updated as following assuming T_I is the most recent BP task:

$$\Delta_j = \frac{\sum_{k=I+1}^{j} N_k}{D_j - D_I}, \text{for } I+1 \le j \le K \qquad \text{E 9}$$

Function: Phase_3_Find_Optimal_DVFS_Policy
Input: *TSet*, f_{sch_min}, X
Output: the energy-optimal $Y=(f_1, f_2, \ldots, f_K)$
{ Calulate $R_1, R_2, \ldots, R_K$; $j = 1$;
while ($j <= K$)
{ Calculate the set $\Delta = (\Delta_j, \ldots, \Delta_K)$
{ $D_i = D - \left(\sum_{k=i+1}^{K} \frac{N_k}{f_{max}} + \frac{XR_i}{f_{max}}\right)$;
$\Delta_i = \frac{\sum_{k=j}^{i} N_k}{D_i}$, for $j \le i \le K$;
}
Find I that $\Delta_I > \Delta_k$ for $I < k \le K$ and $\Delta_I \ge \Delta_k$ for $j \le k < I$;
if ($\Delta_I \le f_{sch\_min}$)
{ let $f_j = \ldots = f_K = f_{sch\_min}$; **break**; }
else
{ let $f_j = \ldots = f_I = \Delta_I$; $D = D - D_I$;
$j = I + 1$;
}
}
}

Figure 30. Find the optimal DVFS policy

Figure 30 shows the pseudo code that finds all the BP tasks and their frequencies. The complexity of finding I is same as that of a search algorithm, $O(K)$. Therefore, the complexity of this algorithm is $O(K^2)$.

3.6.2. Convert the optimal policy to the discrete system

If the frequency of task T_i is between two frequency levels, i.e. $f^l \geq f_i > f^{l+1}$, T_i will be assigned the higher frequency f^l. This guarantees feasibility. Next, there might be some room to decrease the frequencies of some tasks without violating feasibility. The tasks are chosen according to a simple heuristic – the task that has the largest $N_i \cdot \frac{dE(f)}{df}|_{f=f_i}$. Denote this value as the Energy Gradient (***EG***) of task T_i at frequency f_i. The algorithm attempts to reduce the frequency of the task with the largest *EG* to its next lower level. The algorithm uses a max-heap maintain the *EG* of all tasks at their current levels.

Hence such attempt is done by extracting the task at the root of heap. The attempt is successful if the new policy is feasible. And the *EG* of the extracted task at the lower level is inserted back to the heap. Otherwise, the task is discarded and the task with the second largest *EG* (again at the root of the heap) will be considered. The complexity of one such iteration is $O(lg(K))$. The search stops when no task can run at lower frequencies. In the worst case, the algorithm stops when the frequency levels of at most K-1 tasks are reduced from f_{max} to f_{min} which gives a complexity $O(L \cdot K)$. Therefore the complexity of this algorithm is $O(L \cdot K \cdot lg(K))$. The actual runtime is much less since the input of this algorithm is already close to the optimal policy. Finally, Phase 3 will examine all the tuples of (X^l, f^l) from Phase 1. Therefore the complexity of Phase 3 is $O(L^2 \cdot K \cdot lg(K)+K^2)$. Overall the proposed approach including all phases has a complexity $O(L^2 \cdot K \cdot lg(K)+K^2)$.

Table 2. The processors used in simulation

Processor	Pentium M 765	Opteron	Athlon64-4000+	PXA255	Marvell XScale
Performance States (Frequency, Voltage)	(2.1GHz, 1.340V) (1.8GHz, 1.276V) (1.6GHz, 1.228V) (1.4GHz, 1.180V) (1.2GHz, 1.132V) (1.0GHz, 1.084V) (0.8GHz, 1.036V) (0.6GHz, 0.988V)	(2.8GHz, 1.40V) (2.6GHz, 1.35V) (2.4GHz, 1.30V) (2.2GHz, 1.25V) (2.0GHz, 1.20V) (1.8GHz, 1.15V) (1.0GHz, 1.10V)	(2.4GHz, 1.40V) (2.2GHz, 1.35V) (2.0GHz, 1.30V) (1.8GHz, 1.25V) (1.0GHz, 1.10V)	(400MHz, 1.65V) (300MHz, 1.43V) (200MHz, 1.32V)	(1GHz, 2.05V) (800MHz, 1.65V) (600HMz, 1.3V) (400MHz, 0.99V) (200MHz, 0.7V)

3.7. Evaluation

To evaluate the proposed three-phase approach on energy consumption and runtime saving, we developed five processor simulators based on the published data of Intel Pentium M 765, Intel PXA255, AMD Opteron, AMD Athlon64 4000+, Marvell XScale. All of them support discrete DVFS [75][76][77][78][86]. The specifications of these processors are listed in Table 2. A performance state is a voltage/frequency pair.

The first experiment is to evaluate the accuracy of the reliability bound proposed in Phase 1. We randomly create task sets that each one comprises 100 tasks and each task has a cycle number (N) between 0.5 million and 20 million. λ_0 and d are set to 10^{-6} and 4 respectively according to [56]. Given a set of consecutive frequency levels, each task set is assigned 10 random DVFS policies that use the frequencies in the set. For each policy, the reliability of a DVFS policy is calculated by exhaustively accumulating the probabilities of all fault patterns, and the lower bound is calculated according to [73]. Table 3 shows the average difference between them. A row of this table shows the difference for policies using different numbers of levels, and a column shows the differences for a policy with different fault numbers. We choose the Pentium M 765 processor simulator because it supports up to 8 frequency levels.

Table 3 Difference between the reliability lower bound and its actual value

The number of faults	The range of the frequency levels used in a policy							
	1	2	3	4	5	6	7	8
$X = 1$	9.57e-14	6.64e-12	1.16e-10	2.15e-9	4.30e-8	9.66e-7	2.65e-5	1.11e-3
$X = 2$	9.49e-20	4.55e-17	3.01e-15	2.15e-13	1.70e-11	1.56e-9	1.83e-7	3.49e-5

As expected, the difference reduces with the increase of X, and increases with the number of frequency levels used in a policy. Since the proposed technique uses the lower bound of reliability to estimate the reliability of a policy, the reliability of a produced schedule always meets or exceeds R_G. The inaccuracy becomes significant when the range of levels *used in a policy* (except the re-execution frequency f_{max}) is beyond 5, which will result in pessimistic schedules. Thanks to the convex relation between energy and

frequency, the optimal DVFS policy tends to assign tasks the same frequency or a small set of consecutive frequencies, which helps confine the inaccuracy of the proposed lower bound.

```
Function: Exhaustive Search
Input: R_G, K, {N_1, N_2, …, N_K}, L, {f^1 = f_max, f^2, …, f^L = f_min} ;
Output: the optimal policy Y;
{   Find_Optimal_Order (TSet);
    YSet = {Y_i} contains the complete list of policies;
    for each policy Y_i in YSet
    {   if the energy of Y_i ≥ Current_Minimum_Energy
            Break;
        Reliability = 0;
        for (x=0; x≤MaxX; x++)
        {   if Y_i is not feasible Break;
            Reliability += P(x,Y_i);
            if Reliability ≥ R_G
                Current_Minimum_Energy = energy of Y_i ;
        }
    }
}
```

Figure 31 Exhaustive search

The second experiment is designed to evaluate the quality of the complete package of the proposed approach. The uniqueness of the proposed technique lies on the following points: it addresses uni-core systems, it addresses the adverse effect of voltage scaling on fault susceptibility using a fast heuristic; it takes system-level reliability as the only reliability metric, it exams the optimal execution order. To the best of our knowledge, there is no other technique that addresses the same issue. For example, the technique proposed in [61] addresses multi-core systems and requires both a traditional fault tolerance constraint and a reliability goal. The technique proposed in [87] uses task-level reliability and a continuous voltage scaling model. The continuous model makes the problem a convex optimization problem while the discrete model used this work makes the problem a MCKP problem. As a result, we

choose to compare the result of our work to the optimal solution produced by exhaustive searches. The pseudo code is shown in Figure 31. *Note that because the exhaustive search is independent to all the approximations and assumptions made in the proposed three-phase approach, it provides a fair evaluation of the complete package.* For each processor listed in Table 2 we randomly generated 350 task sets with different numbers of edges in the data dependency graph and utilizations of normal tasks execution ranging from 10% to 70%. Utilizations higher than 0.7 are not tested because the reliability goal may not be satisfied. The system-level reliability goals R_G are set to 0.999 999 99 and 0.999 999 9 respectively, according to [61]. P_{idle} and P_{ind} are set to be 0.1w. All the other parameters are the same as the first experiment. Energy consumption is calculated based on the model given in [62]. The simulations are performed in a PC with two 3.4GHz CPUs, 2GB RAM.

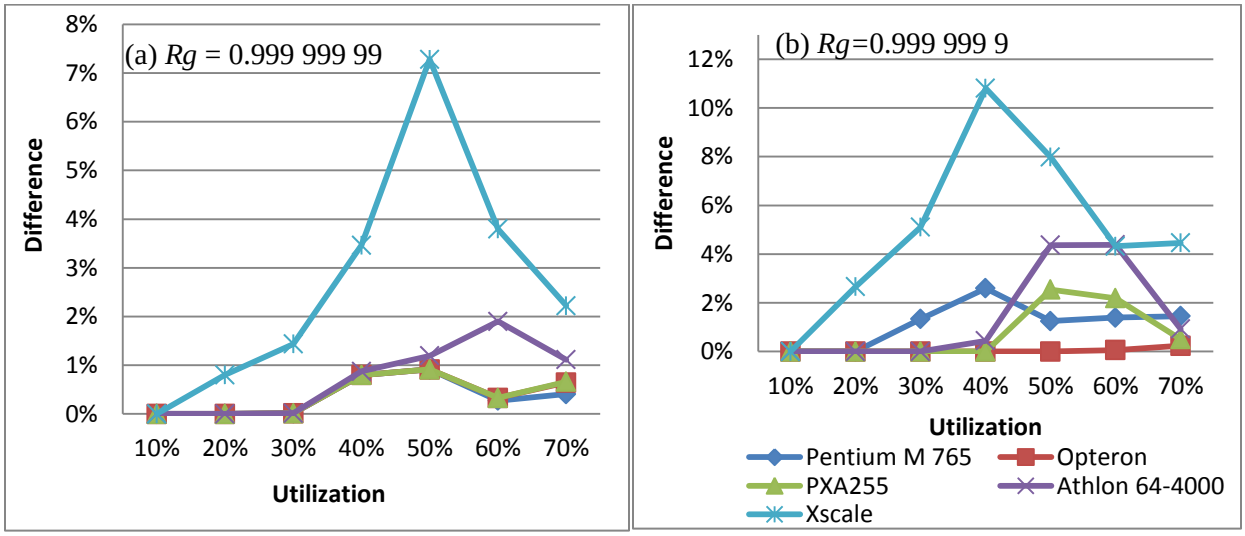


Figure 32 The comparison between the energy consumed by the DVFS policies produced by the proposed technique and the energy consumed by the optimal DVFS policies

Comparison results are reported in Figure 32 with (a) for R_G = 0.999 999 99 and (b) for R_G = 0.999 999 9. In these figures, *x*-axis is the utilization and *y*-axis is the energy difference. As one may see the energy difference is usually less than 5% except the Marvell processor. The reason of the large difference of Marvell processor is due to its large steps between its performance states. As shown in Table 2, the difference between consecutive states of Marvell processors is about 0.3 to 0.4v while the corresponding

difference of other processors is about 0.05 to 0.2v. Hence a small difference in DVFS policy could result in a large difference in energy consumption. Another observation is that the difference peaks when the utilization is between 0.3-0.6. This is because when the utilization is low (< 0.4), all tasks tend to be scheduled on or near f_{Ymin}. Hence difference between exhaustive search and the proposed search is smaller. The reason is also applied to the high utilization (>0.6) where all tasks tend to be scheduled on or near $f_{\max}$. But because tasks' frequencies are high, a small difference in DVFS policy may lead to a large difference in energy consumption. The most error appears at the mid-utilization region. This is because at this region tasks' frequencies spread across a wider spectrum. Therefore the chance of two approaches producing different policies is higher.

Speed wise, the runtime of the proposed approach is on the order of milliseconds. No noticeable runtime increase is observed for the task sets consisting of up to 1000 tasks. The exhaustive search on the other hand spends several minutes for small-scale task sets that comprise up-to 14 tasks. Its runtime increases dramatically as the number of tasks and/or the number of frequency levels increase.

3.8. Conclusions

This section studies the trade-off among energy efficiency, fault tolerance, and reliability in scheduling real-time tasks. Being aware that the problem is NP-hard, we propose to solve this problem in three phases: the first phase converts the reliability goal to a list of conventional fault tolerance constraints, the second phase determines the execution order, and the third phase derives the optimal schedule for each constraint in the list. The final schedule is the one that consumes the least energy. The proposed approach finds a close-to-optimal schedule for a set of K tasks scheduled on a system supporting L voltage levels in $O(L^2 \times K^2)$ time. The difference between the results of the proposed approach and the optimal results is small.

Chapter 4. Security concern of crypto devices with scan chains[5]

4.1. Motivation

Radio frequency identification (RFID) is an automated data capture technology that can be used to electronically identify, track, and store information contained on a tag that is attached to an object, such as a product, case, or pallet. The main technology components of an RFID system are a tag, reader, and database. A reader scans the tag for data and sends the information to a database, which stores the data contained on the tag as shown in Figure 33. RFID tags are made up of a microchip with some data storage and an antenna. Tag readers broadcast an RF signal to access information stored on the tags. RFIDs are an important cross-section technology whose potential application can be found in practically all areas of daily life and business. However, unprotected tags may pose a serious threat on information security and consumer privacy since the communication between tags and readers can be easily eavesdropped [106]. The potential risks include consumer location privacy tracking, sales figure tracking, industrial espionage, unauthorized access to the tag's memory, and counterfeit of tags [89].

Figure 33 Components of an RFID system

Many issues regarding information security within RFID systems can be solved by some cryptographic technology. Among all the modern cryptographic technologies, stream cipher encryption is an ideal option due to the proven security, low hardware and power cost, and high efficiency. For example, the Mifare RFID chips, which are used in many mass transit systems worldwide including Boston, London, and Netherland, encrypt the tag ID using a stream cipher called Crypto-1[103]. However, even if stream cipher encryption is used, RFID chips may still suffer from many advanced attacks. A trio of MIT

[5] Copyright © ACM, 2011. This chapter is a minor revision of the work published in ACM Transactions on Design Automation of Electronic Systems, Volume 16, Issue 2, March 2011, http://doi.acm.org/10.1145/1929943.1929952

students has discovered the security weaknesses of Mifare card used in the Boston public transportation system, and showed how to hack the RFID chip to get free subway rides [102]. In this chapter, we present a general attack that targets a group of stream ciphers, including one of the candidates of the latest ECRYPT eSTREAM project [92].

4.2. Introduction

Stream cipher is an important class of encryption algorithm. They encrypt plaintext messages one bit at a time, in contrast to block ciphers that operate on large blocks of data. Consequently, stream ciphers have simple hardware circuitry, are generally faster and consume very low power. Stream ciphers are deployed in applications where buffering is limited or characters are processed individually such as in wireless telecommunications applications. Stream ciphers have limited or no error propagation and hence are advantageous in noisy environments where transmission errors are highly probable. A stream cipher encrypts plaintext bits using a pseudorandom keystream. As shown in Figure 34 (a), one bit of plaintext is combined with one bit of keystream at a time, typically using a simple XOR operation. Thus, the security of a stream cipher depends on the randomness of its keystream. In practice, the pseudorandom keystream is generated by the keystream generator using a user provided secret key.

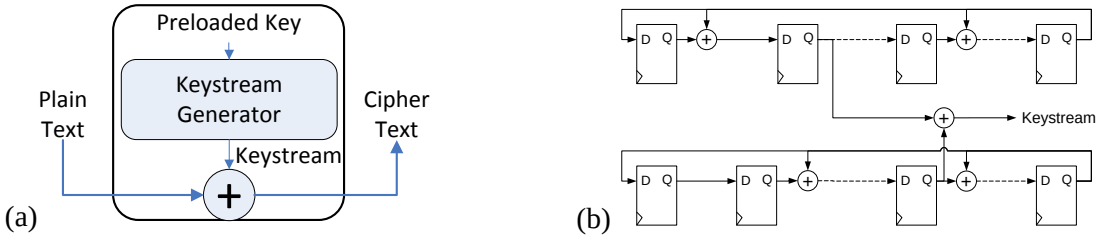


Figure 34: (a) Stream Cipher (b) An example LFSR based keystream generator

In this chapter, we assume that the user key is preloaded into the device, which is the typical scenario in practice. The keystream generator is implemented either as a Linear Feedback Shift Register (LFSR) or other methods such as a linear congruence generator. Due to ease of implementation, long period, and

uniformly distributed output, LFSR based stream ciphers are quite popular. An example keystream generator that uses two LFSRs is shown in Figure 34 (b).

Scan-based Design-For-Test (DFT) is one of the most popular methods to test IC devices. Scan-based DFT ties all flip-flops (FFs) in one or more scan chains, and the states of the FFs can be scanned out through these chains as shown in Figure 35. Scan-based DFT provides access to the internal state of (crypto) hardware by improving control of internal nodes from the primary inputs and observation of values on internal nodes at the primary outputs.

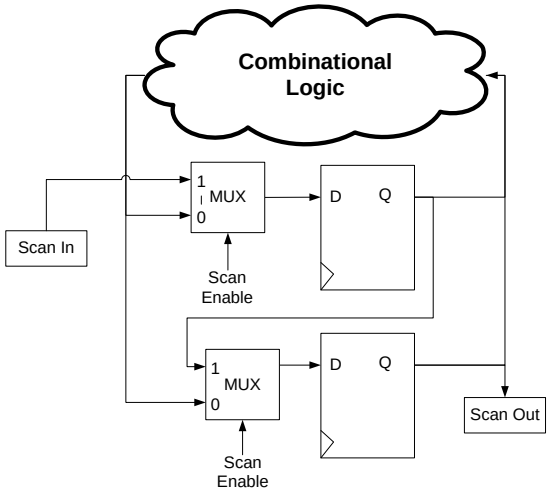


Figure 35: The structure of a scan chain

A flip-flop included in a scan chain is replaced by a scan flip-flop (a D flip-flop with a MUX at the D input). During normal mode when the Scan_Enable signal is set to 0 the scan flip-flop works like a regular D flip-flop. In test mode when the Scan_Enable signal is set to 1 scan flip-flops are disconnected from the combinational circuit and connected as a scan chain. Now, the flip-flop contents can be set to predetermined values (controllability) and intermediate state can be scanned out (observability). Scan chains are typically inserted into the design by test synthesis tools. A scan chain is classically organized according to the physical positions of the flip-flops. Even if they are arbitrarily connected it does not improve the security as shown in this chapter. During chip packaging, scan chains are either connected to the external JTAG interface for in-field debug and maintenance [98] or left unbound to prevent access. In

the former case, it is easy to compromise the secrets on the chip. In the latter case, unbound scan chains can still be accessed [109].

While scan-based DFT improves the quality of testing, it also opens a powerful side channel to the privacy information stored in the design under test, and hence, weakens the theoretically guaranteed security of cipher algorithms [Goering 2004]. Until now, scan chains in DES and AES block cipher implementations have been exploited to leak their secret keys [108][109] as follows:

- First, the positions of scan elements in the scan chain are determined. For this, pairs of known plaintexts that are different in a single bit position are applied in the normal mode and then the internal state of the hardware implementation is scanned out in the test mode.
- Then the secret key is discovered. Modules in the block cipher are analyzed to identify a class of plaintexts. By applying a small number of plaintexts from this special class in the normal mode and by scanning out the corresponding internal state in the test mode, the secret keys are discovered.

In this chapter, we propose a scan-based attack on the hardware implementations of LFSR-based stream ciphers that use scan-based DFT. In contrast to the scan attacks of block ciphers, the proposed scan attack of stream ciphers does not require the attacker to apply carefully designed plaintexts. It is also worth noticing that hardware designers not realizing the security implications of scan DFT may use it in security devices [94].

We will introduce the general technique to determine the scan chain structure of several types of LFSR structures used in stream ciphers in section 4.4 and follow it up by demonstrating the attack on six stream ciphers DECIM [90], Pomaranch [96], A5/1, A5/2 [91], w7 [105], and LILI II [90].

4.3. General description of the attack

We assume the attacker has physical access to the Device-Under-Attack (DUA). An attacker can temporarily access a victim RFID tag or RFID-based toll card as shown in Figure 33, launch the attack

and return the tag or toll card without being noticed. With the obtained secret keys, the attacker can eavesdrop on the victim's communication or clone the card for unauthorized usage. Specifically, we assume that the attacker

- Knows the algorithmic details of the stream cipher since they are public;
- Can run the DUA for a certain number of clock cycles in its normal mode without being noticed;
- Can scan out the states of internal registers of DUA via scan chains after each clock cycle;
- Does not need to scan in special inputs in the test mode and does not need to apply chosen inputs to the stream cipher. This makes the proposed attack different from and more powerful than the one proposed in [107].

After each scan out operation, the attacker will obtain a bit vector that includes all bits of the LFSR and all bits of the architectural registers (AR). ARs are the registers that are not in the cipher specification but in the DUA implementation. This is because a cipher generally provides only the algorithm-related specifications. Different DUAs of the same cipher may have minor differences. For example, different state coding may introduce different numbers of state registers. *Since LFSRs are initialized by the secret key and an initial vector, a stream-cipher-based DUA can be reproduced if the initial states of all the LFSRs are recovered even though the actual secret key itself may not be known. The goal of the attacker is to discover the correspondence between the bits of the scan-out vector and the bits in the LFSRs in the stream cipher*.

The attacker scans out the internal registers at the time when the DUA is initialized and records the scan-out vector V_0. He then resets the DUA, clocks it by one cycle, and records the new scan-out vector as V_1. The attacker repeats this procedure for a certain number of rounds and uses all the recorded vectors to reconstruct the state information of the DUA. The attack can be vital.

4.4. Scan attack on LFSR based stream ciphers

We will describe several attacks that target general LFSR structures. The attack on a specific stream

cipher is a combination of some or all of these attacks. We will analyze the case where the scan-out vector consists of bits of the LFSRs and the ARs. The states of the ARs are assumed to be random. Let *N* be the length of the scan-out vector and *L* be the length of the LFSR, $N \geq L$. Then *N-L* is the number of FFs in the ARs. The following notations are used in this chapter.

LFSR Linear Feedback Shift Register

AR Architectural Register

MNR Maximum number of rounds needed to determine a flip-flop in a search

LBB Left boundary bit of all the discovered bits

RBB Right boundary bit of the discovered bits

N The length of the scan-out vector

L The length of the LFSR

4.4.1. Scan attack on external (Fibonacci) LFSR based stream ciphers

Figure 36 shows two *L*-bit external LFSRs with *N-L* bits ARs. One LFSR has an input and the other has no input. Since the attacks on both are similar, we will only illustrate the attack using the external LFSR with no input.

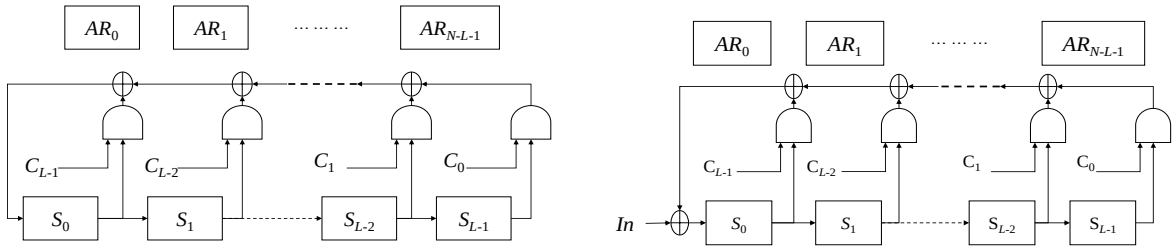


Figure 36: An *L*-bit external LFSR (a) without an input and (b) with an input

The bits in an external LFSR without an input have the **Update Functions**:

$$S_i(t) = S_{i-1}(t\text{-}1) \quad \text{for } 1 \le i \le L\text{-}1 \tag{1}$$

$$S_0(t) = \sum_{0 \le i \le L-1} (C_i \times S_{L-1-i}(t\text{-}1)),\ C_i = 0 \text{ or } 1 \tag{2}$$

The Update Functions show how $S_i(t)$ at clock cycle t is updated by the values from time t-1. $S_i(t)$ is the state of the i^{th} stage at clock cycle t (scanned out as part of the vector V_t). $S_{i-1}(t\text{-}1)$ is the state of the $(i\text{-}1)^{th}$ stage at cycle t-1 (scanned out as part of the vector V_{t-1}). C_i ($1 \le i \le L$-1), $S_i(t)$ and $S_{i-1}(t\text{-}1)$ could be 1 or 0 depending on the characteristic polynomial of the LFSR. To discover the bit-by-bit correspondence between the scan-out vector and the flip-flops in the LFSR, the attacker randomly picks a bit X from one of the scan-out vectors, and checks if X belongs to the LFSR by performing an α-search.

α-search is of two types. For a bit X, the **left α-search** looks for another bit W where $W(t\text{-}1)=X(t)$, while a **right α-search** looks for another bit Y where $X(t)=Y(t+1)$.

(a)

Vector	1	2	3	4	5	6	7	8	9	10	Suspect set of W
V_0	0	0	0	0	0	0	0	1	1	0	All bits except 9
V_1	1	0	0	0	0	0	0	0	0	0	1, 2, 3, 4, 5, 6, 7, 10
V_2	0	1	0	0	0	0	0	0	0	1	2, 3, 4, 5, 6, 7, 10
V_3	0	0	1	0	0	0	0	0	1	1	2, 10
V_4	1	0	0	1	0	0	0	0	0	0	2
V_5	0	1	0	0	1	0	0	0	0	1	2
V_6	0	0	1	0	0	1	0	0	1	0	2
V_7	1	0	0	1	0	0	1	0	1	1	Miss

(b)

Vector	1	2	3	4	5	6	7	8	9	10	Suspect set of W
V_0	0	0	0	0	0	0	0	1	1	0	All bits except 8
V_1	1	0	0	0	0	0	0	0	0	0	1, 2, 3, 4, 5, 6, 7, 10
V_2	0	1	0	0	0	0	0	0	0	1	2, 3, 4, 5, 6, 7, 10
V_3	0	0	1	0	0	0	0	0	1	1	3, 4, 5, 6, 7
V_4	1	0	0	1	0	0	0	0	0	0	4, 5, 6, 7
V_5	0	1	0	0	1	0	0	0	0	1	5, 6, 7
V_6	0	0	1	0	0	1	0	0	1	0	6, 7
V_7	1	0	0	1	0	0	1	0	1	1	7

Figure 37 Left α-search is either (a) successful (i.e. hit) or (b) not successful (i.e. miss)

Let us consider an 8-bit external LFSR with feedback polynomial $1+x^3+x^8$. Its update functions are $S_0(t) = S_2(t\text{-}1)+ S_7(t\text{-}1)$, and $S_i(t) = S_{i-1}(t\text{-}1)$ for $1 \le i \le 7$. For clarity, we assume that the bits of the scan-out vectors are in the same sequence as the bits in the LFSR, i.e. the 1^{st} bit is S_0, the 8^{th} bit is S_7, and the 9^{th} and 10^{th} bits are ARs. Hence N = 10 and L=8. The left α-search on this example is shown in Figure 37 (a) where the 9^{th} bit of the scan-out vector is chosen to be X. The attacker finds its left neighbor W by starting with a "suspect set" which initially contains all bits except X. The attacker prunes this suspect set by eliminating those bits whose values in V_i are different from the value of X in V_{i+1}. This is one round of

checking. If the suspect set is emptied after several rounds of checking as shown in Figure 37 (a), the search is unsuccessful in finding a left neighbor and is hence called a "**miss**". The attacker then randomly picks another bit and repeats this procedure. In Figure 37 (b) S_7 is determined to be the left neighbor W of S_8 after 7 rounds of checking, and the search returns a "**hit**".

Since the bits in the LFSR are pseudorandom and the bits in the ARs are assumed random, each bit in the scan-out vector has a 50% chance to be 1 or 0. If a bit is not the W of the current X, it will be denoted as a **false bit**, and the probability of eliminating this false bit from the suspect set at the n^{th} round equals

$$\prod_{1 \le t \le n-1} Pb_{W(t-1)=X(t)} \times Pb_{W(n-1)\neq X(n)} = 0.5^n$$

where $Pb_{W(t-1)=X(t)}$ is the probability that $W(t\text{-}1)=X(t)$. Therefore, the probability of eliminating a false bit from the suspect set after n rounds equals

$$\sum_{1 \le t \le n} 0.5^n = 1 - 0.5^n$$

If the chosen X is a bit of the LFSR and has the expected Update Function $W(t\text{-}1)=X(t)$, its suspect set will never be emptied. Otherwise, its suspect set will be emptied eventually after certain rounds of searching. Hence the attacker needs to determine the maximum number of rounds (MNR) in the way that a "**hit**" is deemed if the set contains just one bit after so many rounds of searching. Given the sizes of today's LFSR-based stream ciphers, we define the MNR as the number of rounds after which more than 99.99% false bits are eliminated. The MNR of α-search for this example and for all the stream ciphers is thus 15.

A hit during left α-search discovers two bits, X, and its left neighbor W, in the LFSR. A bit is the Left Boundary Bit (LBB) if its left neighbor is undiscovered. A bit is the Right Boundary Bit (RBB) if its right neighbor is undiscovered. After one successful left α-search, W is the LBB and X is the RBB. In the example of Figure 37 (b), bit 7 is the LBB and bit 8 is the RBB. Repeatedly applying the left α-search on LBB will discover the LFSR bits all the way to its left-most bit S_0. Repeatedly applying the right α-search on an RBB will discover all bits in the LFSR all the way to the right most bit S_{L-1}. At this point, the

structure of the LFSR is identified. It is important to note that the *SAME* set of scan-out vectors can be analyzed to discover multiple bits in the LFSR. Therefore for each stream cipher, the attacker only needs to scan out MNR+1 internal state vectors.

4.4.2. Scan attack on internal (Galois) LFSR based stream ciphers

Structure of an internal LFSR in some stream ciphers is shown in Figure 38. The bits in an internal LFSR have the following relations:

$$S_i(t) = S_{i-1}(t\text{-}1) \oplus (C_i \cdot S_{L-1}(t\text{-}1)) \quad (1 \le i \le L\text{-}1), \tag{3}$$

$$S_0(t) = S_{L-1}(t\text{-}1) \tag{4}$$

C_i ($1 \le i \le L$-1) could be 1 or 0 depending on the characteristic polynomial of the LFSR.

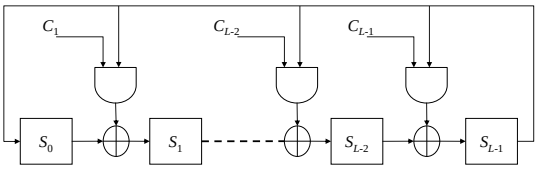


Figure 38 An *L*-bit internal LFSR

When $C_i = 1$, $S_i(t) = S_{i-1}(t\text{-}1) \oplus S_{L-1}(t\text{-}1)$, and is referred to as a tap bit. When $C_i = 0$, there is no feedback involved. $S_i(t) = S_{i-1}(t\text{-}1)$ ($1 \le i \le L$-1), and is referred to as a non-tap bit. The non-tap bits can be discovered by the α-search. Discovering and identifying the tap-bits needs a new type of search, named β-search.

In the left β-search, for a selected bit *X*, a pair of bits in the scan-out vector (*W*, *Z*) are found such that $W(t\text{-}1)=X(t)$ when $Z(t\text{-}1) = 0$, or $W(t\text{-}1) = X(t)$' when $Z(t\text{-}1) = 1$.

Similarly, in the right β-search, for a selected bit *X*, a pair of bits in the scan-out vector (*W*, *Z*) are found such that $X(t) = Y(t+1)$ when $Z(t) = 0$, or $X(t) = Y(t+1)$' when $Z(t) = 1$.

The β-search is based on the observation that $S_i(t) = S_{i-1}(t\text{-}1)$ or $S_i(t) = S_{i-1}(t\text{-}1)$' when $S_{L-1}(t\text{-}1) = 0$ or 1 respectively. For the first β-search, the attacker has to guess two bits, the neighbor bit of X and S_{L-1}. The number of possible 2-tuples for bit X is $P(U, 2) = U(U\text{-}1)$, where U is the number of undiscovered bits in the LFSR. However, when the first β-search returns a hit, bit S_{L-1} is identified. To discover the remaining bits using β-search, the attacker needs to guess only one bit. This reduces the suspect set of 2-tuples to $P(U, 1) = U$. The probability of eliminating a 2-tuple from the suspect set in the n^{th} round equals:

$$\prod_{1 \le t \le n-1} (Pb_{Z(t-1)=1} \times Pb_{W(t-1)=X(t)'} + Pb_{Z(t-1)=0} \times Pb_{W(t-1)=X(t)}) \times [Pb_{W(n-1) \neq X(n)'} \times Pb_{Z(n-1)=1} + Pb_{W(n-1) \neq X(n)} \times Pb_{Z(n-1)=0}] = 0.5^n.$$

More than 99.99% false 2-tuples are eliminated from the suspect set in 15 rounds (MNR). The attack procedure that combines α and β searches is as follows:

1) The attacker randomly picks a bit and applies the α-leftward-search. If the search returns a miss, the attacker randomly picks another bit and continues with α-leftward-searches until he gets a hit. The hit discovers a non-tap bit and its left neighbor which are the RBB and LBB of the discovered bits respectively.

2) The attacker applies the α-leftward-search on LBB and the α-rightward-search on RBB. This step is repeated to grow the discovered section until either the whole LFSR is discovered (LBB = RBB), or both the leftward and the rightward searches return misses. The latter case indicates that the attacker has reached tap-bits.

3) The attacker applies the β-leftward-search on the LBB and the β-rightward-search on the RBB to locate the left neighbor of LBB and right neighbor of RBB, which will become the new LBB and RBB respectively. Then go to step 2. Steps 2) and 3) are repeated until all the bits in the LFSR are discovered. Since the first β-search identifies bit S_{L-1}, this bit can be used to identify all the other tap bits in this LFSR.

4.4.3. Scan attack on internal LFSRs with inputs

An internal LFSR with input is the same as the one shown in Figure 38 except that S_{L-1} is XORed with the input before feedback to the previous stages. The bits have the following relations where *In* stands for the input:

$$S_i(t) = S_{i-1}(t\text{-}1) \oplus (C_i \cdot (S_{L-1}(t\text{-}1) \oplus In)),\ 1 \le i \le L\text{-}1 \quad (5)$$

$$S_0(t) = S_{L-1}(t\text{-}1) \oplus In \quad (6)$$

C_i ($1 \le i \le L$-1) could be 1 or 0 depending on the characteristic polynomial of the LFSR. When $C_i = 0$, no feedback is involved and $S_i(t) = S_{i-1}(t\text{-}1)$ ($1 \le i \le L$-1). These non-tap bits can be discovered by α-searches. When $C_i = 1$, $S_i(t) = S_{i-1}(t\text{-}1) \oplus S_0(t\text{-}1) \oplus In$. Determining these tap bits, however, requires extra effort since *In* is not accessible by the attacker. The attacker considers two tap bits, LBB S_i and the RBB S_j at the same time. Since

$S_i(t) = S_{i-1}(t\text{-}1) \oplus S_0(t\text{-}1) \oplus In$ and $S_{j+1}(t) = S_j(t\text{-}1) \oplus S_0(t\text{-}1) \oplus In$, and

$$S_i(t) \oplus S_{j+1}(t) = S_{i-1}(t\text{-}1) \oplus S_0(t\text{-}1) \oplus In \oplus S_j(t\text{-}1) \oplus S_0(t\text{-}1) \oplus In = S_{i-1}(t\text{-}1) \oplus S_j(t\text{-}1)$$

A new γ-search for such a discovery is defined as follows:

γ-search: Given a pair of bits *X* and *F*, this search looks for a 2-tuple (*W*, *G*) where $W(t\text{-}1) \oplus F(t\text{-}1) = X(t) \oplus G(t)$.

The attacker has to guess two bits *W* and *G*, therefore the suspect set of 2-tuples for bits *X* and *F* is P(*U*, 2) where *U* is the number of undiscovered bits. The probability of eliminating a 2-tuple from the suspect set at the n^{th} round is:

$$\left(\prod_{1 \le t \le n\text{-}1} Pb_{W(t\text{-}1) \oplus F(t\text{-}1) = X(t) \oplus G(t)}\right) \times Pb_{W(n\text{-}1) \oplus F(n\text{-}1) \ne X(n) \oplus G(n)} = 0.5^n$$

More than 99.99% false 2-tuples can be eliminated in 15 rounds (MNR). The attack procedure combining the α and γ searches is similar to the one combining α and β searches. Just replace β with γ in step 3.

4.4.4. Scan attack on LFSRs with jump registers

A modified jump cell, has been proposed to replace the normal register cell used in LFSRs in [97]. Figure 39 shows a register cell that can work in two modes controlled by the Jump Control switch. When the switch is open, it works as a normal register cell and when the switch is closed, it works as a jump cell. Clocking an LFSR using normal cells J times produces the same result as clocking once the same LFSR that uses jump cells instead. J is the jump index derived from the characteristic polynomial.

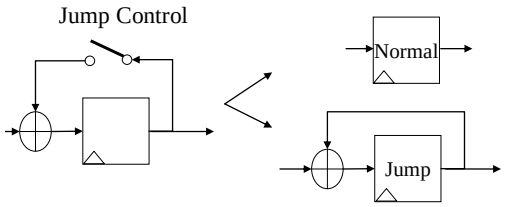


Figure 39 A Jump Cell

Attacking an LFSR using jump cells does not require any new type of search. However, the number of rounds to eliminate a bit/tuple from the suspect set increases. The bits in an external LFSR using jump cells have the following relations where *JC* stands for Jump Control:

$$S_i(t) = S_{i-1}(t\text{-}1) \oplus JC \cdot S_i(t\text{-}1), \text{ for } 1 \le i \le L\text{-}1, \quad (7)$$

$$S_0(t) = \sum_{0 \le i \le L-1} (C_i \cdot S_{L-1-i}(t\text{-}1)) \oplus JC \cdot S_0(t\text{-}1),\ C_i = 0 \text{ or } 1 \quad (8)$$

Observe that when $S_i(t\text{-}1) = 0$, $S_i(t) = S_{i-1}(t\text{-}1)$. The jump version of α-search defined below can discover the bits in an external LFSR using jump cells:

α-leftward-search-J: Given a bit X, it looks for another bit W where $W(t\text{-}1) = X(t)$ when $X(t\text{-}1) = 0$.

α-rightward-search-J: Given a bit X, it looks for another bit Y where $X(t\text{-}1) = Y(t)$ when $Y(t\text{-}1) = 0$.

The probability of eliminating a bit from the suspect set at the n^{th} round is

$$\prod_{1 \le t \le n-1} (Pb_{X(t-1)=1} + Pb_{X(t-1)=0} \times Pb_{W(t-1)=X(t)}) \times (Pb_{X(n-1)=0} \times Pb_{W(n-1)\neq X(n)}) = (3/4)^{n-1} \times 1/4$$

More than 99.99% false bits are eliminated from the suspect set within 33 rounds (MNR). The MNR to eliminate more than 99.99% tuples from the suspect set is doubled. The attack procedure for the LFSRs using normal cells can also be applied to the LFSRs using jump cells.

4.4.5. Scan attack on irregular clock controlled LFSRs

Irregularly stepping the LFSR through successive states is a method to increase the linear complexity of an LFSR while preserving a large period and good statistical properties. Stream ciphers based on regularly clocked LFSRs are susceptible to basic and fast correlation attacks [93][104]. Irregular clocking limits the possibilities for mounting classical correlation attacks.

For devices using irregular clocked LFSRs, consecutive scan-out vectors could be partly or completely the same. These redundant vectors should be abandoned. To determine if a vector is redundant, the attacker can check if the bits of interest to the search (i.e. the (X W) in an α-search, the (X W Z) in a β-search, and the (X W F G) in a γ-search) have different values from the bits in the previous vector. The MNR to eliminate the bits/tuples from a suspect set increases inversely with the probability of the LFSR being clocked in a cycle.

4.4.6. Scan attacks on stream ciphers with multiple LFSRs

Most LFSR-based stream ciphers use more than one LFSR. To determine the scan chain structure, we need to locate all the LFSRs. Therefore, we need to establish a one to one correspondence between the LFSRs discovered and the LFSRs in the cipher. If the LFSRs have different lengths, we can easily get the mapping. If the LFSRs have the same length, we can still map them according to their unique feedback functions.

4.5. Putting it all together: Attacks on selected LFSR-based stream ciphers

4.5.1. DECIM

DECIM is a stream cipher submitted to the ECRYPT eSTREAM project [92]. It uses an 80-bit key, a 64-bit IV, and a 192-bit external LFSR. The key stream generation mechanism is shown in Figure 40. The bits of the external LFSR are numbered from 0 to 191. The Boolean function f is a 13-variable quadratic symmetric function. ABSG is an irregular decimation mechanism. DECIM uses a 32-bit key buffer to maintain a constant throughput for the key stream. The LFSR is regularly clocked.

The α-search is sufficient to attack DECIM. While we estimate that the MNR = 15 is sufficient for an α-search, we use MNR = 17 in our simulations to ensure a high level of confidence. The total number of checking performed is $17 \times N = 17 \times 228 = 3876$.

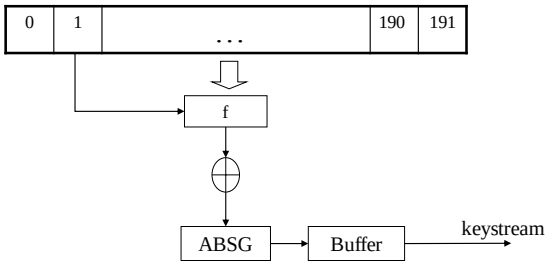


Figure 40 DECIM Stream Cipher

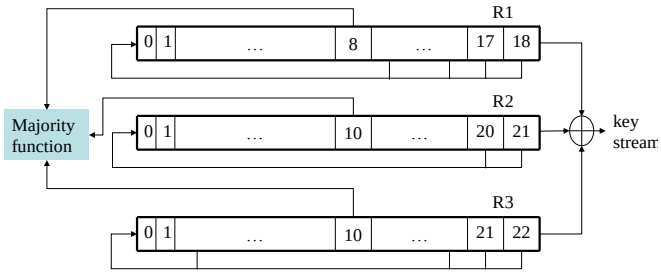


Figure 41 A5/1 Stream Cipher

4.5.2. A5/1

A5/1 is a stream cipher used to encrypt over the air transmissions in the GSM standard. A GSM conversation is transmitted as a sequence of 228-bit frames (114 bits in each direction) every 4.6 milliseconds. To ensure privacy, each frame is XORed with a 228-bit keystream produced by A5/1. As shown in Figure 41, A5/1 cipher uses three external LFSRs – R1, R2, and R3 of lengths 19, 22, and 23 bits, respectively. At each cycle, after the initialization phase, the left-most bits of the LFSRs are XORed to produce one bit key. The three LFSRs are irregularly clocked depending on the output of a majority function M. M computes the majority of S_8 of R1, S_{10} of R2 and S_{10} of R3. An LFSR shifts only when the state of its selected bit equals M.

To attack A5/1, we need to apply α-searches to determine the three register segments. After that, we can tell them apart by their lengths. The number of vectors used by our simulation is 32 (ie. MNR =31) and the total number of comparisons is about 31×N=31×64=1984.

4.5.3. A5/2

A5/2 is a stream cipher used to provide voice privacy in the GSM cellular telephone protocol. A5/2 uses four external LFSRs R1, R2, R3 and R4 of lengths 19, 22, 23, and 17 bits respectively as shown in Figure 42. Clocking of R1, R2 and R3 is controlled by R4 and R4 is regularly clocked in each clock cycle. A majority function is attached to an LFSR and outputs the majority of three selected bits from the LFSR. The outputs of all the majority functions and the right most bit from each register are XORed to produce the output.

The procedure to attack A5/2 is basically the same as that of A5/1. The number of scan-out vectors used in our simulation is 42 (i.e MNR = 41) and the total number of comparisons is approximately 41×81 = 3321.

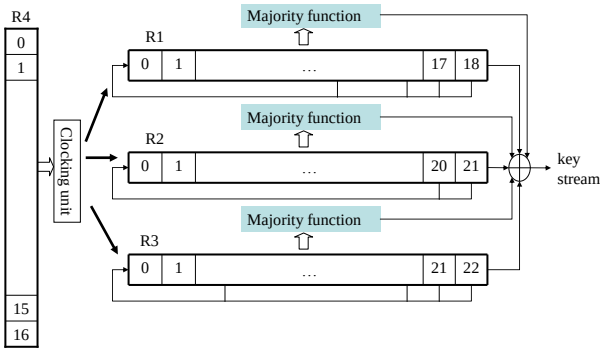


Figure 42: A5/2 Stream Cipher

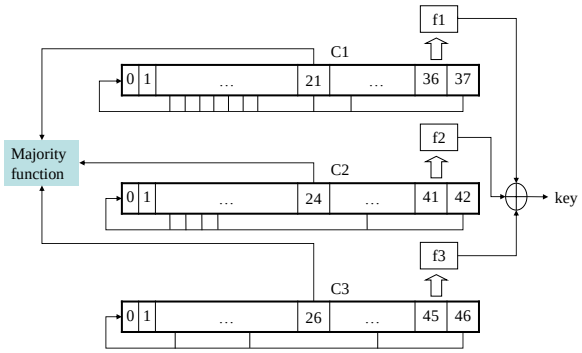


Figure 43: A module in W7 stream cipher

4.5.4. W7

W7 is a synchronous stream cipher optimized for efficient hardware implementation [105]. W7 cipher contains eight similar modules each of which consists of three external LFSRs and one majority function as shown in Figure 43. The majority function in a module controls the clocking of the LFSRs in the

module and the clocking principle is the same as that of A5/1. The outputs of all modules compose a byte of the key stream.

Since all the LFSRs are external, applying α-search is sufficient to discover the bits of all the LFSRs. The identity of a LFSR can be told by matching the unique lengths and feedback functions of discovered LFSRs. The MNR used in our simulation is 83 and the total number of comparisons is $83 \times N = 83 \times 1024 = 84992$.

4.5.5. LILI II

LILI-II is a simple and fast stream cipher that uses two internal LFSRs. As shown in Figure 44, LILI II has two subsystems: one subsystem generates an irregular clock to control the other subsystem that produces the keystream. The LFSR in the clock-control subsystem is regularly clocked. The Fc function in the system takes the first bit in the LFSR S_0 and the 127^{th} bit in the LFSR S_{126} and computes $Fc = 2S_0 + S_{126} + 1$. Since the output of Fc could be 1, 2, 3, or 4, the LFSR in the keystream generation subsystem is clocked 1, 2, 3, or 4 times respectively between two consecutive key bits.

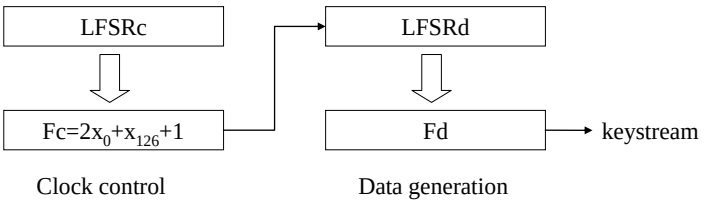


Figure 44: LILI II Stream Cipher

Since both LFSRs are internal, applying α-search and β-search can discover all bits. The identities of the two LFSRs can be told by matching the lengths. The MNR used by our simulation is 62 and the number of checking is $62 \times N = 62 \times 255 = 15810$.

4.5.6. Pomaranch

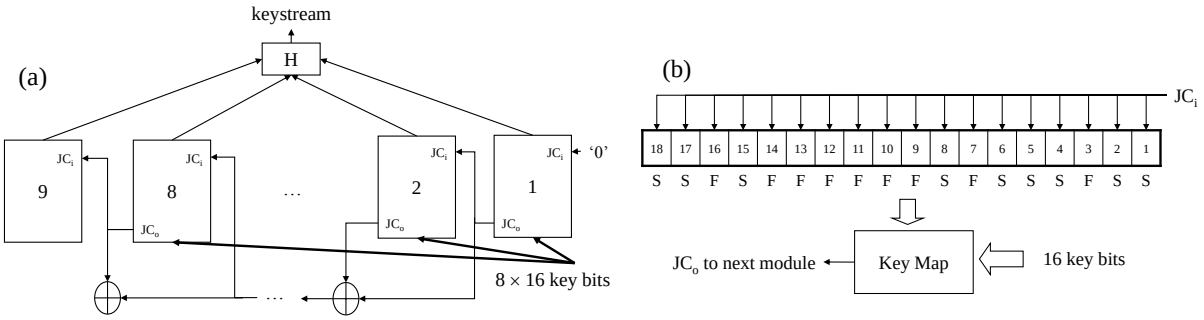


Figure 45: Pomaranch stream cipher (a) Top level (b) The odd module

The keystream generator in Pomaranch is called the cascade jump controlled sequence generator consisting of 9 modules as shown in Figure 45(a). Each module has an 18-bit shift register using F and S cells as shown in Figure 45(b). An F cell works as a jump cell when JC_i of this module is 1, or as a regular shift cell when JC_i is 0. An S cell works as a jump cell when JC_i of this module is 0, or as a regular shift cell when JC_i is 1. All cells are regularly clocked. The inputs to the key map module comprise of 9 bits from the LFSR and 16 bits of the secret key. The 1-bit output of the Key Map is sent to the next module as the JC_i. For the 9 modules, all the even numbered modules share a configuration of F and S cells and a feedback function, while all the odd numbered modules share another configuration of F and S cells and another feedback function. The attack steps are as follows:

1) Discover the 9 LFSRs by applying the jump version of α-search.

2) Identify if a discovered LFSR belongs to an odd module or an even module by matching its feedback function.

3) Since the JC_i of the first module is always 0, it will match the discovered LFSR whose F and S cells never switch modes.

4) For the remaining modules, the cell working modes depend on the JC_i of the module that in turn depends on the LFSR and the 16-bit key used by the previous module. Since the LFSR of the first module is identified in step 3), we can guess JC_i by simulating every possible 16-bit key (from 0x0000 up to 0xFFFF) and see if it agrees with the cell working modes of any discovered LFSR that belongs to an even

module.

5) Repeat step 4 above to attack the remaining modules.

Overall, the MNR used in our simulation is 42 and the total number of checking is $42 \times N = 42 \times 162 = 6804$.

Table 4 summarizes the simulation results of all the ciphers we attack. Note that the number of scan-out vectors needed to launch such scan-based attack is just MNR + 1. The total number of comparisons equals MNR × N, which takes negligible time in a modern computer.

Table 4 The summary of the stream ciphers

Cipher	LFSR Type	MNR	N	Clock Control
DECIM	External	17	228	Regular
A5/1	External	32	64	Irregular
A5/2	External	41	81	Irregular
W7	External	83	1024	Irregular
LILI II	Internal	62	255	Irregular
Pomaranch	Jump	42	162	Regular

4.6. The state-of-art countermeasures

Not all DFT techniques may introduce security vulnerabilities. For example, in large chips and processors with over tens of thousands of flip-flops, test data is typically compressed on chip. This adds an additional layer of security that prevents the attacker from recovering the bit-by-bit information of the scan chains – the attacker would have to work on compressed scan-out vectors. However, in embedded processors and crypto accelerators used in low-end smart cards, test data is not compressed owing to the limited number of flip-flops. With debug becoming mandatory, test inputs and test outputs are not compressed even in some large processors [98]. Therefore the countermeasure that ensures security while maintaining

testability is of great interests. However, the state-of-art countermeasures either bring additional security concerns or are not cost-effective when stream ciphers are considered, as we explain below.

The first countermeasure against scan-based attacks is a scan-chain scrambling technique [95]. The scrambling technique partitions a scan chain into multiple segments. The connections between segments are via MUXs and can be altered by giving different control signals to these MUXs. If a tester fails authentication, the segments will be connected in an unpredictable way so that the bits in a scan-out vector do not correspond to the bits in a different scan-out vector. Clearly, the effectiveness of such technique relies on the effectiveness of tester authentication. However, this important issue is not addressed by the authors.

Another authentication-based secure-scan architecture called “Lock & Key” is proposed in [98]. Similar to the scan-scrambling technique, the “Lock & Key” technique also partitions the scan chains into multiple equal-length segments. The test security controller examines the test key that is input through the scan path. If the provided key passes the authentication, the LFSR in the controller will be seeded by the seed provided by the tester so the scan in/out operation is predictable. Otherwise, the LFSR will be seeded randomly and the scan operation is not predictable. Another secure-scan architecture called LCSS is processed in [100]. LCSS is again an authentication-based technique. The Key Checking Logic (KCL), which is a K-input-1-output combinational logic, examines the key provided through the scan path, and will zero-out the scan-out vectors if an invalid key is detected. It can be seen that the security of both techniques depends on the privacy of the test key. Since all chips produced in a batch will share the same test key, how to keep the test key private among different consumers is a significant security concern, especially when chip fabrications are usually outsourced to a third-party foundry nowadays. A makeup scheme is to implement the key checking circuit using re-configurable logic and allows end users to re-configure test keys. But the significant cost at both manufacturing phase and run-time is in conflict with the primary reason to use stream ciphers where efficiency and low-cost are greatly appreciated.

A secure scan architecture that does not depend on tester authentication is shown in Figure 46 [107]. The private key used by the crypto core is stored in the KEY register that does not join any scan chain. This technique defines two modes for CUT: Secure mode and Insecure mode. In Insecure mode, the tester can scan in any key to the Mirror Key Register (MKR) for the purpose of testing or debugging. The real private key is not involved in Insecure mode therefore scanning out the internal registers does not leak its secret. In Secure mode, the private key stored in the KEY register is loaded into the Mirror Key Register for normal operations. Transition from Secure mode to Insecure mode is always followed by a global reset. However in the case that a stream cipher is deployed, since the initial status of the LFSR is the secret, employing a Mirror Key Register basically doubles the number of FFs in the circuit.

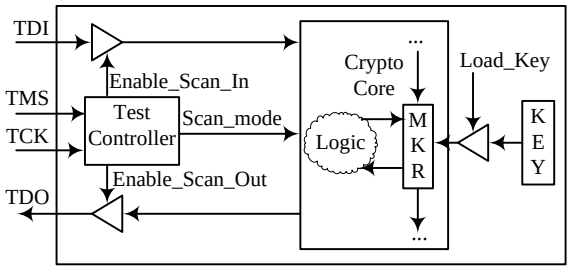


Figure 46: MKR based secure scan architecture

4.7. Conclusion

In this chapter we propose a new scan attack that targets a group of LFSR-based stream ciphers. The attack analyzes the scan-out vectors to discover the internal states of DUA. The number of scan-out vectors required is less than 20 for some ciphers and is less than 100 for all the ciphers attacked in our research. The CPU time for processing the vectors and identifying the bits of LFSR is negligible. With the knowledge of the LFSRs used in stream cipher devices, an attacker can clone an authentication device, eavesdrop a private conversation, etc. The state-of-art countermeasures either bring additional security concerns, or are not cost-effective when stream ciphers are considered. This calls for new cost-effective secure scan architectures for stream ciphers. If a cost-effective secure scan architecture or built-in self-test is used, then this approach will not be successful.

Chapter 5. Future work

Power and energy efficient error detection techniques at circuit and system levels are presented in this thesis. As we have shown, unification of fault security and power efficiency is an interesting and challenging topic. In the high level synthesis approaches, there are three major challenges. First, when the schedule of a design is fixed, finding the power optimal binding is NP-hard. Second, simultaneous scheduling and binding for power minimization is NP-hard. Third, we prove that when scheduling and binding are done, inserting the minimum number of checkpoints for fault security is NP-hard. The proposed iterative improvement approach can solve fairly large inputs with 50 operation of each type in duplicated DFG. Even though the runtime has been improved significantly compared to the recent works that handles at most 20 operations and do not consider fault security constraints, it is very time consuming to handle DFGs that has 50+ operations for a given type. One bottle neck is the ILP formulation to compute the optimal power. Future work can focus on how to solve this problem more efficiently. Once the complexity is reduced, more high level synthesis tasks, such as register allocation, can be considered.

For the system level technique, we present a task scheduling algorithm for single processor frame based systems. The technique optimizes the energy consumption under the system deadline constraint and reliability constraint. There are two major limitations in this technique. First, it is designed for single processor system. Extension to multi-processor system would be useful. For example, a task assignment algorithm can be developed to assign the tasks to different processors so that our single processor algorithm can be applied afterwards. Another limitation is the frame based system model. In the frame based system model, all tasks are released at time 0 of a time frame and should be finished before the end of the frame. Though frame based system has received a lot of attention, to have general applicability to real-time systems, it is useful to extend the technique to handle task sets where each task has an individual release time and deadline.

CITED LITERATURE

[1] Ramesh Karri, Alex Orailoglu: High-Level Synthesis of Fault-Secure Microarchitectures. DAC 1993: 429-433.

[2] Ganesh Lakshminarayana, Anand Raghunathan, Niraj K. Jha: Behavioral Synthesis of Fault Secure Controller/Datapaths Based on Aliasing Probability Analysis. IEEE Trans. Computers 49(9): 865-885, 2000.

[3] Kaijie Wu, Ramesh Karri, Algorithm Level Re-Computing - A Register Transfer Level Concurrent Error Detection Technique, ICCAD 2001: 537-543.

[4] Kaijie Wu, Ramesh Karri: Fault secure datapath synthesis using hybrid time and hardware redundancy. IEEE Trans. on CAD of Integrated Circuits and Systems 23(10): 1476-1485, 2004.

[5] Anand Raghunathan, Niraj K. Jha: An ILP Formulation for Low Power Based on Minimizing Switched Capacitance During Data Path Allocation. ISCAS 1995: 1069-1073.

[6] Janak H. Patel, Leona Y. Fung: Concurrent Error Detection in ALU's by Recomputing with Shifted Operands. IEEE Trans. Computers 31(7): 589-595, 1982

[7] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev R. Rao, Toan Pham, Conrad H. Ziesler, David Blaauw, Todd M. Austin, Krisztián Flautner, Trevor N. Mudge: Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. MICRO 2003: 7-18

[8] Balaji Narasimham, Bharat L. Bhuva, Ronald D. Schrimpf, Lloyd W. Massengill, Matthew J. Gadlage, Oluwole A. Amusan, William Timothy Holman, Arthur F. Witulski, William H. Robinson, Jeffrey D. Black, Joseph M. Benedetto, Paul H. Eaton, Characterization of digital single event transient pulse-widths in 130-nm and 90-nm cmos technologies, IEEE Trans. Nuclear Science, 54(6): 2506-2511, 2007.

[9] Yiorgos Makris, Ismet Bayraktaroglu, Alex Orailoglu: Invariance-Based On-Line Test for RTL Controller-Datapath Circuits. VTS 2000: 459-464.

CITED LITERATURE (Continued)

[10] Natasa Miskov-Zivanov, Kai-Chiang Wu, Diana Marculescu: Process variability-aware transient fault modeling and analysis. ICCAD 2008: 685-690

[11] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadladge, T. Turflinger: Digital Single Event Transient Trends With Technology Node Scaling, IEEE Trans. Nuclear Science, 53(6): 3462-3465, 2006

[12] Noureddine Chabini, Wayne Wolf: Unification of scheduling, binding, and retiming to reduce power consumption under timings and resources constraints. IEEE Trans. VLSI Syst. 13(10): 1113-1126, 2005

[13] Lisa M. Guerra, Miodrag Potkonjak, Jan M. Rabaey: High Level Synthesis for Reconfigurable Datapath Structures. ICCAD 1993: 26-29

[14] Lars Kruse, Eike Schmidt, Gerd Jochens, Ansgar Stammermann, Arne Schulz, Enrico Macii, Wolfgang Nebel: Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs. IEEE Trans. VLSI Syst. 9(1): 3-14, 2001

[15] Noureddine Chabini, Ismaïl Chabini, El Mostapha Aboulhamid, Yvon Savaria: Methods for minimizing dynamic power consumption in synchronous designs with multiple supply voltages. IEEE Trans. on CAD of Integrated Circuits and Systems 22(3): 346-351, 2003.

[16] Saraju P. Mohanty, N. Ranganathan, Sunil K. Chappidi: Simultaneous peak and average power minimization during datapath scheduling for DSP processors. ACM Great Lakes Symposium on VLSI 2003: 215-220

[17] J. C. Lu, B. Taskin: Clock buffer polarity assignment with skew tuning. ACM Trans. Design Autom. Electr. Syst. 16(4): 49, 2011.

[18] Kaijie Wu, Ramesh Karri: Fault secure datapath synthesis using hybrid time and hardware redundancy. IEEE Trans. on CAD of Integrated Circuits and Systems 23(10): 1476-1485, 2004

[19] Anne Mignotte, Olivier Peyran: Reducing the Complexity of ILP Formulations for Synthesis. ISSS 1997: 58-64

CITED LITERATURE (Continued)

[20] In-Cheol Park, Chong-Min Kyung: FAMOS: an efficient scheduling algorithm for high-level synthesis. IEEE Trans. on CAD of Integrated Circuits and Systems 12(10): 1437-1448, 1993

[21] Anand Raghunathan, Niraj K. Jha: An iterative improvement algorithm for low power data path synthesis. ICCAD 1995: 597-602

[22] Anand Raghunathan, Niraj K. Jha: SCALP: an iterative-improvement-based low-power data path synthesis system. IEEE Trans. on CAD of Integrated Circuits and Systems 16(11): 1260-1277, 1997

[23] Ganesh Lakshminarayana, Anand Raghunathan, Niraj K. Jha: Behavioral Synthesis of Fault Secure Controller/Datapaths Based on Aliasing Probability Analysis. IEEE Trans. Computers 49(9): 865-885, 2000

[24] Chun-Gi Lyuh, Taewhan Kim: High-level synthesis for low power based on network flow method. IEEE Trans. VLSI Syst. 11(3): 364-375, 2003

[25] B. W. Kernighan, S. Lin: An Efficient Heuristic Procedure for Partitioning Graphs, Bell Sys. Tech. 49(2): 291-308, 1970.

[26] S. Lin and B. W. Kernighan: An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Operations Research, 21(2): 498-516, 1973.

[27] H. K. Kwan: A multi-output second-order digital filter structure for. VLSI implementation. IEEE Trans. Circuits Syst. CAS-32: 108-109, 1985.

[28] Pierre G. Paulin, John P. Knight, Emil F. Girczyc: HAL: a multi-paradigm approach to automatic data path synthesis. DAC 1986: 263-270

[29] S. J. Orfanidis, Introduction to Signal Processing. Upper Saddle River, NJ: Prentice-Hall, 1996.

[30] Qian Ding, Rong Luo, Yuan Xie: Impact of process variation on soft error vulnerability for nanometer VLSI circuits. ASIC 2005: 1117-1121.

[31] V. Gerousis and D. Desharnais: Challenges at the 45-nm node are great. EE Times, 2007.

CITED LITERATURE (Continued)

[32] Rajeev R. Rao, Kaviraj Chopra, David T. Blaauw, Dennis Sylvester: Computing the Soft Error Rate of a Combinational Logic Circuit Using Parameterized Descriptors. IEEE Trans. on CAD of Integrated Circuits and Systems 26(3): 468-479, 2007.

[33] Krishnan Ramakrishnan, R. Rajaraman, Sivaprakasam Suresh, Narayanan Vijaykrishnan, Yuan Xie, Mary Jane Irwin: Variation Impact on SER of Combinational Circuits. ISQED 2007: 911-916.

[34] Abhijit Chatterjee, Manuel A. d'Abreu: The Design of Fault-Tolerant Linear Digital State Variable Systems: Theory and Techniques. IEEE Trans. Computers 42(7): 794-808, 1993.

[35] Sudip Chakrabarti, Abhijit Chatterjee: On-line fault detection in DSP circuits using extrapolated checksums with minimal test points. ITC 1999: 955-963.

[36] Ahmad Abdelhay, Emmanuel Simeu: Analytical Redundancy Based Approach for Concurrent Fault Detection in Linear Digital Systems. IOLTW 2000: 112-117.

[37] Maryam Ashouei, Soumendu Bhattacharya, Abhijit Chatterjee: Design of Soft Error Resilient Linear Digital Filters Using Checksum-Based Probabilistic Error Correction. VTS 2006: 208-213.

[38] Maryam Ashouei, Soumendu Bhattacharya, Abhijit Chatterjee: Probabilistic Compensation for Digital Filters Using Pervasive Noise-Induced Operator Errors. VTS 2007: 125-130.

[39] Kaijie Wu, Ramesh Karri: Algorithm level recomputing using allocation diversity: a registertransfer level approach to time redundancy-based concurrent errordetection. IEEE Trans. on CAD of Integrated Circuits and Systems 21(9): 1077-1087, 2002

[40] Ismet Bayraktaroglu, Alex Orailoglu: Accumulation-based concurrent fault detection for linear digital state variable systems. Asian Test Symposium 2000: 484-488.

[41] Ismet Bayraktaroglu, Alex Orailoglu: Cost effective digital filter design for concurrent test. International Conference on Acoustic, Speech and Signal Processing 2000: 3323-3326.

[42] Ismet Bayraktaroglu, Alex Orailoglu: Unifying methodologies for high fault coverage concurrent and off-line test of digital filters. ISCAS 2000: 705-708.

[43] U. Sparmann: On the Check Base Selection Problem for Fast Adders. VTS 1993: 62-65.

CITED LITERATURE (Continued)

[44] Issam Alzaher-Noufal, Michael Nicolaidis: A CAD Framework for Generating Self-Checking 1 Multipliers Based on Residue Codes. DATE 1999: 122-129.

[45] Balaji Narasimham, Bharat L. Bhuva, Ronald D. Schrimpf, Lloyd W. Massengill, Matthew J. Gadlage, Oluwole A. Amusan, William Timothy Holman, Arthur F. Witulski, William H. Robinson, Jeffrey D. Black, Joseph M. Benedetto, Paul H. Eaton, Characterization of digital single event transient pulse-widths in 130-nm and 90-nm cmos technologies, IEEE Trans. Nuclear Science, 54(6): 2506-2511, 2007.

[46] Yu Liu, Kaijie Wu: Towards Cool and Reliable Digital Systems: RT Level CED with Runtime Adaptability. ICCD 2010: 528-533.

[47] Algirdas Avizienis: Arithmetic Algorithms for Error-Coded Operands. IEEE Trans. Computers, 22(6): 567-572, 1973.

[48] Stanislaw J. Piestrak: Design of Residue Generators and Multioperand Modular Adders Using Carry-Save Adders. IEEE Trans. Computers 43(1): 68-77, 1994.

[49] V. Gutnik and A. Chandrakasan: An efficient controller for variable supply-voltage low power processing, Symposium on VLSI Circuits: 158-159, 1996.

[50] W. Namgoong, M. Yu, and T. Meng: A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator, IEEE Int. Solid-State Circuits Conf. 1997: 380-381.

[51] E. Normand: Single event upset at ground level, IEEE Trans. on Nuclear Science, 43(6): 2742-2750, 1996.

[52] T. Langley, R. Koga, T. Morris: Single-event effects test results of 512MB SDRAMs, IEEE Radiation Effects Data Workshop 2003: 98-101.

[53] Hermann Kopetz: Why time-triggered architectures will succeed in large hard real-time systems. FTDCS 1995: 2-9.

[54] H. Kopetz, R. Obermaisser: Temporal composability, Computing & Control Engineering Journal, 13(4): 156 – 162, 2002.

CITED LITERATURE (Continued)

[55] R.A. Reed, J. Kinnison, J.C. Pickel, S. Buchner,P.W. Marshall, S. Kniffin, K.A. LaBel: Single-event effects ground testing and on-orbit rate prediction methods: the past, present, and future, IEEE Trans. Nuclear Science, 50(3): 622-634, 2003.

[56] C. Weulersse: DASIE Analytical Version: A Predictive Tool for Neutrons, Protons and Heavy Ions Induced SEU Cross Section, IEEE Trans. Nuclear Science, 53(4) : 1876-1882, 2006.

[57] D. Zhu, R. Melhem, and D. Mosse: The effects of energy management on reliability in real-time embedded systems, ICCAD 2004: 35-40.

[58] Baoxian Zhao, Hakan Aydin, Dakai Zhu: Generalized reliability-oriented energy management for real-time embedded applications. DAC 2011: 381-386.

[59] Jian-Jia Chen, Chin-Fu Kuo: Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms. RTCSA 2007: 28-38.

[60] Tohru Ishihara, Hiroto Yasuura: Voltage scheduling problem for dynamically variable voltage processors. ISLPED 1998: 197-202.

[61] D. Zhu and H. Aydin: Energy management for real-time embedded systems with reliability requirements, ICCAD 2006: 528-534.

[62] Paul Pop, Kåre Harbo Poulsen, Viacheslav Izosimov, Petru Eles: Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. CODES+ISSS 2007: 233-238.

[63] R. Jejurikar, C. Pereira and R. Gupta: Leakage aware dynamic voltage scaling for real-time embedded systems, DAC 2004: 275-280.

[64] R. Melhem, D. Moss´and E.Elnozahy: The interplay of power management and fault recovery in real-time systems, IEEE Trans. on Computers, 53(2): 217-231, 2004.

[65] Dakai Zhu, Rami G. Melhem, Daniel Mossé, Elmootazbellah (Mootaz) Elnozahy: Analysis of an Energy Efficient Optimistic TMR Scheme. ICPADS 2004: 559-568.

[66] Dakai Zhu: Reliability-Aware Dynamic Energy Management in Dependable Embedded Real-Time Systems. IEEE Real Time Technology and Applications Symposium 2006: 397-407.

CITED LITERATURE (Continued)

[67] Tongquan Wei, Piyush Mishra, Kaijie Wu, Han Liang: Online task-scheduling for fault-tolerant low-energy real-time systems. ICCAD 2006: 522-527.

[68] Ping Zhu, Fumin Yang, Gang Tu, Wei Luo: Fault-Tolerant Scheduling for Periodic Tasks based on DVFS. ICYCS 2008: 2186-2191.

[69] Alireza Ejlali, Marcus T. Schmitz, Bashir M. Al-Hashimi, Seyed Ghassem Miremadi, Paul M. Rosinger: Energy efficient SEU-tolerance in DVS-enabled real-time systems through information redundancy. ISLPED 2005: 281-286.

[70] Arun K. Somani, Nitin H. Vaidya: Understanding Fault Tolerance and Reliability - Guest Editors' Indroduction. IEEE Computer 30(4): 45-46, 50, 1997.

[71] Kenneth H. Rosen, Discrete Mathematics and Its Applications, McGraw-Hill Higher Education, 2007.

[72] Pascal Chevochot, Isabelle Puaut: Scheduling Fault-Tolerant Distributed Hard Real-Time Tasks Independently of the Replication Strategies. RTCSA 1999: 356-363.

[73] Han Liang, Energy-efficient fault-tolerant scheduling techniques for dependable real-time systems, ProQuest, 2011

[74] Viacheslav Izosimov, Paul Pop, Petru Eles, Zebo Peng: Design Optimization of Time- and Cost-Constrained Fault-Tolerant Distributed Embedded Systems. DATE 2005: 864-869.

[75] Inki Hong, Gang Qu, Miodrag Potkonjak, Mani B. Srivastava: Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. RTSS 1998: 178-187.

[76] Intel Corporation: Intel Pentium M Processor on 90 nm Process with 2-MB L2 Cache Datasheet, http://www.intel.com, 2006.

[77] Intel Corporation: Intel PXA255 Processor Electrical, Mechanical, and Thermal Specification, http://www.intel.com, 2004.

[78] AMD Corporation: AMD Athlon 64 Processor Power and Thermal Data Sheet, http://www.amd.com, 2006.

CITED LITERATURE (Continued)

[79] AMD Corporation: AMD Opteron™ Processor Power and Thermal Data Sheet, http://www.amd.com, 2006.

[80] Aamer Mahmood, Edward J. McCluskey: Concurrent Error Detection Using Watchdog Processors - A Survey. IEEE Trans. Computers 37(2): 160-174, 1988.

[81] N. Oh, P. P. Shirvani, and E. J. McCluskey: Control flow checking by software signatures. IEEE Trans. on Reliability, 51(2): 111-122, 2002.

[82] Steven K. Reinhardt, Shubhendu S. Mukherjee: Transient fault detection via simultaneous multithreading. ISCA 2000: 25-36.

[83] Alireza Ejlali, Bashir M. Al-Hashimi, Paul M. Rosinger, Seyed Ghassem Miremadi, Luca Benini: Performability/Energy Tradeoff in Error-Control Schemes for On-Chip Networks. IEEE Trans. VLSI Syst. 18(1): 1-14, 2010.

[84] Premkishore Shivakumar, Michael Kistler, Stephen W. Keckler, Doug Burger, Lorenzo Alvisi: Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. DSN 2002: 389-398.

[85] K. Dudzinski, S. Walukiewicz: Exact methods for the knapsack problem and its generalizations, European Journal of Operational Research, 28(1): 3-21, 1987.

[86] F. Frances Yao, Alan J. Demers, Scott Shenker: A Scheduling Model for Reduced CPU Energy. FOCS 1995: 374-382.

[87] Marvell, XScale microarchitecture, www.marvell.com

[88] Baoxian Zhao, Hakan Aydin, Dakai Zhu: Enhanced reliability-aware power management through shared recovery technique. ICCAD 2009: 63-70.

[89] Yu Liu, Kaijie Wu, Ramesh Karri: Scan-based Attacks on LFSR based Stream Ciphers. ACM Trans. Design Autom. Electr. Syst. 16(2): 2011.

[90] Berbain, C., Billet, O., Canteaut, A., Courtois, N., Debraize, B., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., and Sibert, H: DECIM. http://www.ecrypt.eu.org/stream/decimp3.html

CITED LITERATURE (Continued)

[91] Andrew Clark, Ed Dawson, Joanne Fuller, Jovan Dj. Golic, Hoon Jae Lee, William Millan, Sang-Jae Moon, Leone Simpson: The LILI-II Keystream Generator. ACISP 2002: 25-39.

[92] Erguler, I. and Anarim, E.: A Modified Stream Generator for the GSM Encryption Algorithms A5/1 and A5/2. EUSIPCO 2005.

[93] eStream: Stream cipher project of the European Network of Excellence in Cryptology ECRYPT. http://www.ecrypt.eu.org/stream/

[94] Goering, R.: Scan Design Called Portal for Hackers, EE Times. http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=51200146, 2004.

[95] Gurkaynak, F.K., Luethi, P., Bernold, N., Blattmann, R., Goode, V., Marghitola, M., Kaeslin, H., Felber, N., and Fichtner, W: Hardware Evaluation of eSTREAM Candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST, ZK-Crypt. http://www.ecrypt.eu.org/stream/

[96] David Hély, Marie-Lise Flottes, Frédéric Bancel, Bruno Rouzeyre, Nicolas Bérard, Michel Renovell: Scan Design and Secure Chip. IOLTS 2004: 219-226.

[97] Jansen, C.J.A.: Stream cipher design: Make your LFSRs jump! SASC 2004: 94–108.

[98] Jansen, C.J.A., Helleseth, T., and Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher. http://www.ecrypt.eu.org/stream/pomaranchp3.html.

[99] Don Douglas Josephson, Steve Poehhnan, Vincent Govan: Debug methodology for the McKinley processor. ITC 2001: 451-460.

[100] Jeremy Lee, Mohammad Tehranipoor, Chintan Patel, Jim Plusquellic: Securing Scan Design Using Lock and Key Technique. DFT 2005: 51-62.

[101] Jeremy Lee, Mohammad Tehranipoor, Jim Plusquellic: A Low-Cost Solution for Protecting IPs Against Scan-Based Side-Channel Attacks. VTS 2006: 94-99.

[102] Menezes, A., Van Oorschot, P., and Vanstone, S.: Handbook of Applied Cryptography. CRC Press, 1996.

[103] Mills, E.: D-Day for RFID-based transit card systems. CNET News. http://news.cnet.com/8301-1009_3-10059605-83.html?tag=mncol;title, 2008.

CITED LITERATURE (Continued)

[104] Nohl, K., and Plotz, H.: Mifare – Little security despite obscurity. http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html, 2007.

[105] Thomas Siegenthaler: Decrypting a Class of Stream Ciphers Using Ciphertext Only. IEEE Trans. Computers 34(1): 81-85, 1985.

[106] Thomas, S., Anthony, D., Berson, T., and Gong, G: The W7 Stream Cipher Algorithm. Internet Draft, 2002.

[107] Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, Daniel W. Engels: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. SPC 2003: 201-212.

[108] Bo Yang, Kaijie Wu, Ramesh Karri: Secure scan: a design-for-test architecture for crypto chips. DAC 2005: 135-140.

[109] Bo Yang, Kaijie Wu, Ramesh Karri: Secure Scan: A Design-for-Test Architecture for Crypto Chips. IEEE Trans. on CAD of Integrated Circuits and Systems 25(10): 2287-2293, 2006.

VITA

NAME: Yu Liu

EDUCATION: B. S., Electrical Engineering, Dalian University of Technology, China, 2007

M. S., Computer Engineering, University of Illinois, Chicago, USA, 2011

Ph. D., Electrical and Computer Engineering, University of Illinois, Chicago, USA, 2013 (exp.)

TEACHING: Teaching Assistant at Department of Electrical and Computer Engineering, 2007–2011, Courses: ECE 265 Introduction to logic design, ECE 469 Computer Systems Design

PROFESSIONAL SERVICES: Reviewer of IEEE Transactions on Computers

Reviewer of IEEE International Symposium on Circuits and Systems 2011

Reviewer of the Asia Symposium on Quality Electronic Design 2010

Reviewer of journal of Microprocessors and Microsystems - Embedded Hardware Design

PUBLICATIONS: *Conference proceedings*

Yu Liu, Kaijie Wu: Runtime adaptable concurrent error detection for linear digital systems. ICCD 2011: 261-266.

Yu Liu, Kaijie Wu: Towards Cool and Reliable Digital Systems rt level CED Techniques with runtime adaptability. ICCD 2010: 528-533.

Yu Liu, Han Liang, Kaijie Wu: Scheduling for Energy Efficiency and Fault tolerance in Hard real- time systems. DATE 2010: 1444-1449.

Yu Liu, Kaijie Wu: An ILP Formulation to Unify Power Efficiency and Fault Detection at Register-Transfer Level. DFT 2009: 349-357.

Journal Papers

Yu Liu, Kaijie Wu: A Fault Duration and Location Aware CED Technique with Runtime Adaptability. IEEE Trans. VLSI Syst. Accepted for publication.

Yu Liu, Kaijie Wu, Ramesh Karri: Scan-based Attacks on LFSR based Stream Ciphers. ACM Trans. Design Autom. Electr. Syst. 16(2), 2011.