Detecting Undetectable Controller Faults Using Power Analysis

J. Carletta* C. A. Papachristou† M. Nourani‡

Abstract

In systems consisting of interacting datapaths and controllers, the datapaths and controllers are traditionally tested separately by isolating each component from the environment of the system during test. This is not possible when the controller-datapath pair is an embedded system designed as a hard core. This work facilitates the testing of controller-datapath pairs in a truly *integrated* fashion. The key to the approach is a careful examination of the types of gate level stuck-at faults that can occur within the controller. A class of faults that are undetectable in an integrated test by traditional means is identified. These faults create faulty but functional circuits. The effect of these faults on power consumption is explored, and a method based on power analysis is given for detecting these faults. Analysis is given for three example systems.

1 Introduction

This work addresses the problem of testing systems that consist of interacting datapaths and controllers, and facilitates the testing of these controller-datapath pairs in an integrated fashion. Typically, testing of datapaths and controllers is done independently, rather than as two parts of an inseparable pair. However, the separation may not be feasible in embedded system designs where the controller-datapath is a reusable component or core to be integrated in a system-on-chip. Separate testing does not adequately cover the interface between the controller and the datapath.

Few, if any, design tools address the issue of how to test datapath and controller in an integrated way. The main difficulty in integrated testing is the need to propagate controller faults through the datapath for observation at the datapath outputs. In their work on integrating controller and datapath test, Dey et. al. observed that even when the controller and datapath are 100% testable separately, the combination of them has usually much lower coverage. This degradation, in their opinion, occurs due to the correlation and dependency between the control signals[8].

In our recent work [16] we addressed the problem of integrated test, and provided a test synthesis method to allow the controller to be easily tested as part of the integrated system. However, our method required design-for-testability insertion at the controller-datapath interface, and therefore is not appropriate for embedded cores. In our work we came across a new type of system-level redundant fault, originating in the controller, that while having no functional effect yet produces an undesirable power increase during normal system operation. This issue has been the motivation and focus of our present work. The key to our approach is a careful analysis of the system-functionally redundant faults, which are undetectable in any traditional test that treats the pair as an integrated system. What we found is that many of these faults have a significant, measurable effect on dynamic power consumption. We remark here that these faults can not be caught by IDDQ techniques [1], which measure quiescent current.

* Department of Electrical Engineering, University of Akron, Akron, OH 44325.

† Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106.

‡ Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75083.

To the best of our knowledge this is the first work that studies the effect of functionally redundant stuck-at faults on power consumption during normal operation. This is an important issue for low power or portable embedded systems. Our contributions are: first, to provide a thorough analysis of the analog nature of these otherwise digital faults; and, second, to provide a test procedure based on power analysis for detecting these faults and to suggest an approach for actually implementing this procedure in practice.

There are many well-known problems in controller optimization. based on FSM decomposition and synthesis thechniques [9], [2]. A method for controller testing [12] uses test registers to support self-testable controller. A scan-based controller is reported in [6], and a dedicated test controller is discussed in [14]. However, none of these approaches use a unified model to test the controller-datapath pair. Instead, datapath and controller are tested separately in different test sessions. The basic test scheme is similar to what [5] proposed. That is the controller output signals are multiplexed with some or all of the datapath primary outputs, thus, making them directly observable. Observing the controller and datapath faults separately, in general, implies more test time (due to separate test session) and more overhead (due to direct observation of each).

Recently, the effect of controller design on power consumption has been explored in [15]. The work of [3] uses special state assignments to reduce power, while [4] adds some combinational logic to the original controller to avoid inactive state transitions. to block the global clock and effectively "turn off" the inactive components in the datapath.

2 Background

Two approaches can be taken to testing a controller-datapath pair. The first approach is to test the datapath and controller separately by splitting the pair during test. This approach can achieve high fault coverage, and so has an advantage when the controller and datapath are designed separately. However, separate testing does not fully test the interface between controller and datapath. Also, the approach may be infeasible in embedded controllers because it requires a design-for-testability (DFT) insertion at the controller-datapath interface to accomplish the splitting. In particular, it is not possible to modify an embedded module designed as a hard core. In addition, even when it is possible to do the DFT insertion, it may cause an unacceptable increase in the system critical path.

The second approach, shown in Figure 1, is to test the controller-datapath pair in its entirety as an integrated, inseparable unit. This approach is a necessity if the datapath-controller are designed together as a reusable component or core to be embedded in a larger design. This is very important in view of modern trends in VLSI technology, which emphasize the design reusability of embedded systems. One obvious advantage of integrated testing is that it accommodates testing of the controller-datapath interface lines all the way into the datapath; with a separate test, no matter where the interface is split, there is a potential for faults between the split and the datapath that can not be caught during either phase of the separate tests. Previous work outlines how to test a datapath in an integrated test [17]. However, it is much more difficult to test the controller in an integrated test; the effects of controller faults must be propagated *through* the datapath to the datapath outputs for observation. Our earlier work [16] develops a method for testing the

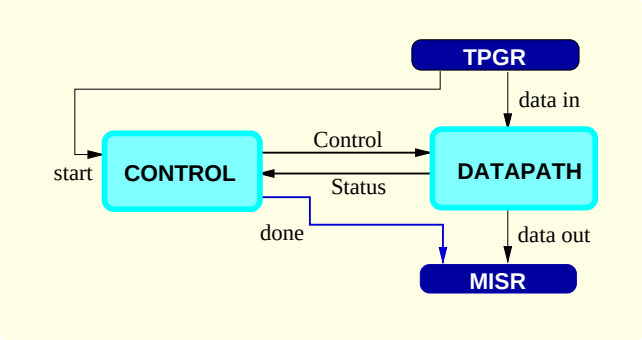


Figure 1: Integrated Controller-Datapath Testing

controller in an integrated scheme, but the approach requires some DFT insertion.

The main difficulty with an integrated test is that even if the datapath and controller are designed so as not to contain redundancy when taken individually, when they are integrated into a controller-datapath pair, redundancies may be introduced. We detail this issue in the next section. Further, we remark that not only are the faults masked by these new system level redundancies difficult to detect, but they also have detrimental effects on the system operation. In particular, they can cause a significant increase in dynamic power consumption. We believe this is an important issue in embedded system design. In what follows we review several fault classes that appear in a controller-datapath pair, as presented in [16].

We classify stuck-at faults internal to the controller (Fig. 2) based on whether a fault affects the functionality of the controller. By functionality, we mean the This means the input-output behavior of the *synthesized* controller as it operates in normal mode. Note that a stuck-at fault within the controller may cause one or more outputs of the controller to change in one or more time steps of the control flow.

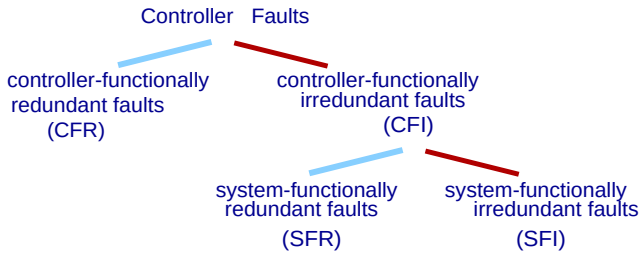


Figure 2: Classification of controller faults.

Controller-functionally redundant or *CFR* faults are faults that never affect the output of the synthesized controller in normal mode. CFR faults can not be caught in either a integrated test or a independent test; they require design-for-testability insertion within the controller itself.

Controller-functionally irredundant or *CFI* faults affect the output of the controller in at least one time step when the controller is running in normal mode. We further divide the CFI faults into two subgroups, based on whether a fault affects the input-output behavior of the controller-datapath pair *as a system*. *System-functionally irredundant* or *SFI* faults are those faults that change the functionality of the system as a whole. It is possible to catch these faults with an integrated test, since a given SFI fault affects the datapath computation for at least some combinations of datapath data inputs. *System-functionally redundant* or *SFR* faults are those faults that do not affect the input-output behavior of the system, even though they did affect the input-output behavior of the controller. It is not possible to detect SFR faults by monitoring the output of the datapath.

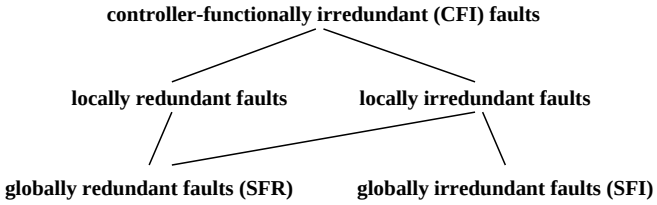


Figure 3: Locally and globally redundant fault classes.

We note that all CFI faults in the controller change the control signals going into the datapath. However, some changes are such that the register transfer level operation of the datapath is not affected. These faults can be considered *locally redundant* in the sense that they at no time change the contents of any register in the datapath from what the contents would be in the fault-free case. Clearly, these faults do not affect the function of the datapath, and are therefore also *globally redundant* or SFR. Still other control line changes cause a change in the contents of some register at some time; these faults can be considered *locally irredundant*. As we shall see in Section 3, a locally irredundant fault may turn out to be either *globally redundant* (SFR) or *globally irredundant* (SFI), depending on whether that local change is propagated to the datapath outputs. This distinction is illustrated in Figure 3.

3 Analysis of SFR Controller Faults

A CFI fault within the controller affects one or more control lines in one or more time steps. It may affect multiplexer select lines, register load lines, or both. We refer to a change in a single control line in a single control step as a *control line effect*. To determine whether a single stuck-at-fault within the controller is SFR, we must look at the interaction of all the control line effects caused by the stuck-at fault.

3.1 Control line effects on select lines

In any given time step, a multiplexer is either *active*, i.e., its output is being used in a computation that will be written to a register, or *inactive*, i.e., its output is discarded.

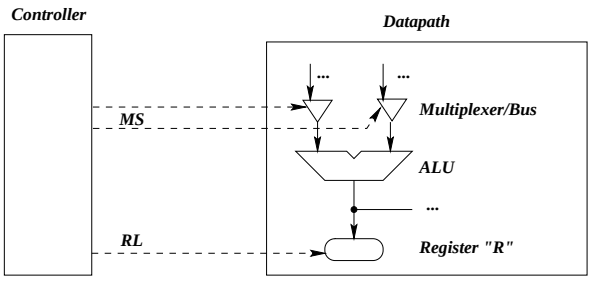


Figure 4: One functional block defining our datapath style.

Referring to Figure 4, if the multiplexer is active, a register at the ALU's output loads the result of the computation at the end of the time step. During time steps when the multiplexer is active, its select lines are "cares" in the sense that any change in a select line will cause an operation to be done on incorrect data, thereby causing a change in the results of the computation. Unless the datapath is itself redundant any control line effect that changes a "care" specification of a multiplexer select line is SFI.

In time steps when the multiplexer is inactive, the corresponding ALU is also inactive, and no register at the ALU's output is loaded. In these time steps, the select lines for the multiplexer are "don't cares". Depending on how the controller was synthesized, the select lines will be either 0s or 1s. The computation result is never written to any register, therefore does not affect the function performed by the datapath. Thus, a control line effect that changes a "don't care" specification on a multiplexer select line is SFR. Whether the fault that caused the control line effect is also SFR depends on whether it also causes another control line effect involving the output register.

3.2 Control line effects on register load lines

Consider a datapath register R in a datapath for which a computation requires T time steps. In general, a number of variables of the data flow are bound to the register, and in any given time step from 1 to T, the register is either *live*, i.e., currently storing some variable from the data flow, or *idle*, i.e., not currently storing a variable. Each variable bound to the register has a *lifespan* starting from the end of time step in which the variable is loaded into the register and ending at the beginning of the time step in which the variable is last read from the register. In Figure 5(a), the register has two variables bound to it, and therefore two live periods or lifespans; the first extends from $LD1$ to $RD2$, and the second extends from $LD2$ to $RD4$. If the datapath is itself not redundant, we can consider that any data stored in a register while the register is live is crucial to the computation.

Control line effects that change register load lines fall into two categories: those that cause the load line to be a 0 in some time step when it should be a 1, thus skipping a load, and those that cause the load line to be a 1 in some time step when it should be a 0, thus causing an extra load. Any control line effect that causes a register load to be skipped is SFI; since the result of a crucial computation is never written, the computation taking place in the datapath is irretrievably disrupted. Thus, any controller fault that causes a control line effect of this type is also SFI.

A control line effect is *disruptive* if it causes a mis-read of some variable in the datapath computation, and *non-disruptive* otherwise. Clearly, a control line effect is *non-disruptive* if it causes an extra load in a time step when the register is idle. For example, $LDf1$ in Figure 5(b) is non-disruptive. Such a control line effect writes garbage data into the register, but does not overwrite a variable, and therefore does not affect the datapath computation. Such a control line effect is SFR.

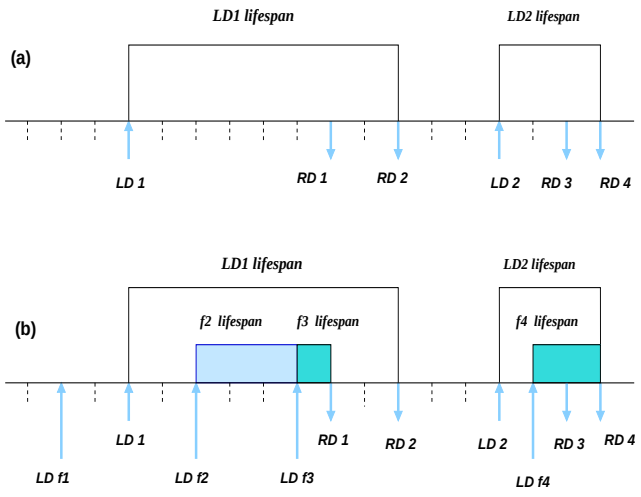


Figure 5: Variable lifespans and register load line fault effects

A control line effect that causes an extra load is *potentially disruptive* if the load occurs within a lifespan of R; in Figure 5(b), $LDf2$, $LDf3$, and $LDf4$ fall into this category. In order to determine whether the effect is disruptive, we have to look at the next read of the register. The control line effect itself has a lifespan that lasts until either the time of the next control line effect that loads that register, or the end of the variable lifespan. When the next control line effect happens, the register is loaded a second time, so the data loaded by the first control line effect is overwritten. When this happens, we say that the first control line effect becomes non-disruptive. This is the case for $LDf2$ in Figure 5(b), which becomes non-disruptive when $LDf3$ arrives. When the variable lifespan ends, the variable that the control line effect disrupts is no longer needed, so the control line effect can have no further effect on the datapath computation.

At the read of the register, we can determine whether a potentially disruptive control line effect is disruptive or non-disruptive. We must look at the specific data involved to see whether the data read is incorrect. This is the case for $LDf3$ in Figure 5(b), which is still potentially disruptive at $RD1$. The data being written to the register can be traced at the register transfer level; note that it depends on the multiplexer select lines, which may also be affected by control line effects. There are two possibilities. For the first case, the extra load writes garbage to the register. The load upsets the datapath computation, because the read references the garbage data. Hence, the potentially disruptive control line effect becomes disruptive. For the second case, the extra load serves simply to rewrite a variable unchanged. Here, the potentially disruptive control line effect becomes non-disruptive.

3.3 Control line effects and SFR faults

In order to determine whether a given fault within the controller is SFR, we must look at the interactions of all the the fault's control line effects. If any one control line effect caused by the fault is SFI, the fault is SFI. If every control line effect caused by a fault is SFR, the fault is SFR. We caution here that one control line effect can change whether another control line effect is disruptive; whether an extra register load is writes in a garbage value depends what is being routed to the register through the multiplexers above it. This in turn may be affected by control line effects on the multiplexer select lines.

4 Power Effects of SFR Faults

Although SFR faults have no effect on system functionality, they may have other detrimental effects in the system operation. In particular, they may cause a significant increase in the system dynamic power consumption. SFR faults affecting select lines will change power consumed in the multiplexers and arithmetic logic units, whereas SFR faults affecting register load lines cause unnecessary loading of unused values, increasing the power consumption of the registers and any combinational logic driven by those registers. In essence, such a fault undermines the gated clock scheme used for low power design. This section uses examples to illustrate the effect of SFR faults on power consumption, referring to Fig. 6.

• **Faults affecting multiplexer select lines**. A control line effect involving a multiplexer select line during time step t is SFR only when the multiplexer select is a "don't care" and the register is not loaded in that time step. In Figure 6, if we assume that x, y, and z do not change between times $t-1$ and t, and that in the fault-free case the multiplexer select line has also been designed to stay at '0' for both time steps, then in the fault-free case there is no energy consumed in the multiplexer and arithmetic logic unit in time step t; the inputs to the combinational logic do not change as time step $t-1$ ends and time step t begins. The block computes $x+z$ throughout both time steps. However, if fault $f1$ acts to change the multiplexer select line to '1', additional energy will be consumed

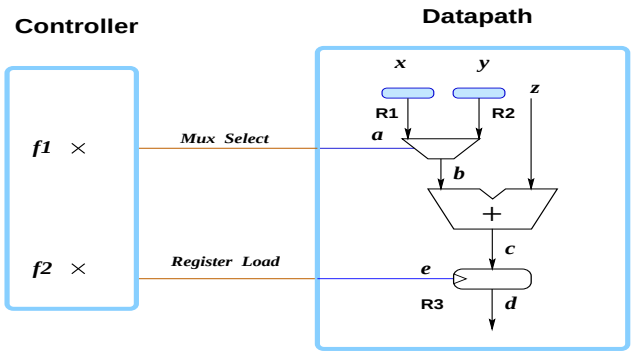


Figure 6: SFR faults affecting multiplexer select and register load lines.

in time step t; the block computes $x + z$ in time step $t - 1$, but computes $y + z$ in time step t.

For real circuits, it is not always the case that x, y and z do not change between time steps $t - 1$ and t; whether they do depends on the register bindings. In addition, the controller may not be designed so that the multiplexer select line stays at '0' throughout the two time steps; while this design choice is best in terms of fault-free power consumption in the datapath, the controller may have been designed without taking power into account. For this reason, we can not guarantee that control line effects on multiplexer select lines always cause an *increase* in power. As we shall see in Section 6, power due to these effects can go either up or down.

• **Faults affecting register load lines**. Again using Figure 6, an SFR fault $f2$ causes a register R_3 to do an extra load in time step t. This load may serve to re-load the same data a second time or may load new data in some way that does not affect the overall operation of the datapath. In either case, the register requires no energy in time-step t in the fault-free case, but requires additional energy in the faulty case. Thus, in the case of SFR faults affecting register load lines, we are guaranteed that power consumption will increase, not only in the affected register, but also in the combinational circuitry driven by that register.

To better illustrate the effect of SFR faults on power consumption, we use an example design that implements a differential equation solver. The design has eleven register load lines, for $REG1$ through $REG11$, and seven multiplexer select lines, $MS1$ through $MS7$. The control flow has 10 states: $CS1$ through $CS8$, plus a RESET state and a HOLD OUTPUT state. The example has a total of 37 SFR faults. We show some representative ones in Table 1. We were careful to choose a faults that show the full range of effect on power for this example, from fault 1, which causes the largest decrease in power, to fault 37, which causes the largest increase. (A graph of all faults can be found in Section 6.) For each fault, we summarize the control line effects caused by the fault, and the change in power consumption for the datapath.

We also experimented by simulating the differential equation solver while adding as many control line effects as possible while still not disrupting the datapath computation. The power increased by over 200% over the fault-free case. While it is highly unlikely that a single stuck-at fault within the controller could cause such an extreme increase in power, this does represent a "worst case" scenario possible with multiple faults.

We have the following observations.

- Although SFR faults are "digital" stuck-at faults within the controller, their only effect on the circuit is of an "analog" nature: a change in power consumption.
- The size of the power change varies. Faults affecting only multiplexer select lines generally have small power effects, up or down. Faults affecting register load lines cause an increase. The increase in power can become quite large if several registers share a load line, or if a fault causes multiple extra loads in different time steps.
- SFR faults can be graded in terms of their importance by experimentally establishing (through simulation) which ones have a large detrimental effect on power. These "power faults" can be detected by a test procedure involving power measurement.

	Control line effects	Power mW	% increase
fault-free	–	1.679	–
fault 1	1. MS3 changes in CS5 2. MS3 changes in CS7	1.628	-3.02%
fault 6	1. MS2 changes in CS3	1.680	0.06%
fault 21	1. REG11 : extra load in CS2 2. REG11 : extra load in HOLD 3. MS3 changes in HOLD	1.722	2.56%
fault 27	1. REG3 : extra load in CS1 2. REG3 : extra load in CS5 3. REG3 : extra load in CS7 4. REG3 : extra load in CS8	1.833	9.17%
fault 37	REG5, 6, 8, and 9 load in all control steps	2.031	20.98%

Table 1: The effect of system-functionally redundant faults on power consumption for 4-bit implementation of a differential equation solver.

5 Detection of SFR Faults that Increase Power

The method for determining whether a particular fault is SFR is based on the analysis of the previous section. However, as a practical matter, we can considerably reduce the number of faults that must be analyzed by using a fault simulation to prove many of the faults system-functionally irredundant (SFI) in a pre-processing step. The following steps are taken:

1. Do a fault simulation of the entire system (datapath and controller) using pseudorandom data generated by a TPGR for the datapath data inputs. Remove any controller faults detected during this simulation from consideration as SFI. For some test pattern generated by the TPGR, the fault causes the datapath to produce an incorrect output.
2. Of the remaining controller faults, remove any faults that are clearly SFI, but were not detected during the simulation due to the limitations of the simulator. For example, the GENTEST simulator we used [10] will mark a stuck-at-0 on the register load line of a primary output register as potentially detected, rather than as detected. When the register load line is stuck-at-0, the register never loads. In the simulation, the register's contents remain unknown throughout the test; therefore, the simulator can not tell whether the contents differ from the fault-free value. However, as a practical matter we know that such a fault will always be detected. In the real circuit, the register will keep whatever value it had at boot-up throughout the test session. Since we expect the register to take on a wide range of values, we know that at some time step there will be a mismatch.
3. For each controller fault that remains, inject the fault into the controller and simulate the controller to determine the fault's effect on the controller outputs. If the fault does not change at least one controller output in at least one time step, the fault is controller-functionally redundant (CFR), and should be removed from consideration.

4. Analyze the remaining controller faults to determine which are SFR, using the method outlined in the previous section.

Once we have determined which controller faults are system-functionally redundant, we can grade their importance based on their effect on power. Of course, power consumption in the datapath depends on the specific data used in the computation. To get an idea of the average power consumption over a wide range of test sets, a Monte Carlo simulation can be used; the faulty circuit is simulated for random data until the power converges. This is reasonable in the absence of knowledge of the kind of data that will be used for a specific application. If a fault has only a small effect on power, or even decreases power, it is of minor importance; the fault is not detrimental to the system's operation. If, on the other hand, the effect on power is large, the fault is an important one. For practical purposes, we must choose a threshold percentage, and say that the fault is important if it causes a percentage change bigger than the threshold.

Before this approach for detecting SFR faults can be practically applied, several difficulties must be overcome. First of all, it must be possible to measure the power attributed to the controller-datapath pair. Testers can monitor power of a chip under test. If the datapath-controller pair is an embedded core, we must somehow separate out its power from the power of the rest of the chip. Power management schemes employed in large microchips can be potentially useful in this case.

The second difficulty is that the threshold must be chosen large enough to accomodate normal variations in a core's power consumption, due to process variations when the chip was fabricated, environmental variations, et cetera. The smaller the threshold can be made in practice, the greater is the percentage of SFR faults that can be detected with this technique.

The third difficulty is that we must be sure that an SFR fault's effect on power is reasonably consistent for different test sets. We must be confident that if a SFR fault is deemed "important" because it significantly increases power in the Monte Carlo simulation, we will be able to easily find a short test set, practical to apply, that also shows a large power increase.

6 Experimental Results

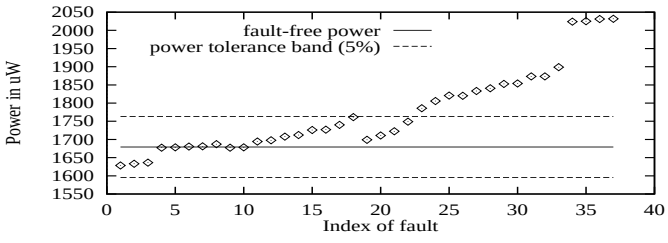
In this section, we demonstrate our approach using three example circuits. The circuits have been synthesized from high level descriptions using the SYNTEST synthesis system [13]. The output of SYNTEST is a register transfer level datapath and state diagram controller. Logic level synthesis is done using the ASIC Synthesizer from the COMPASS Design Automation suite of tools [7], using a finite state machine implementation for the controller and based on a 0.8-micron CMOS library [18]. We could have filled in the controller don't care specifications so as to optimize power in the datapath, but we purposely did not; to do so would have made our scheme look optimistically good, by making all SFR faults cause power increases. All three example circuits have four bit wide datapaths. The first implements a differential equation solver and is a standard high level synthesis benchmark [11]. The second example is another high level benchmark known as the FACET example [11]. The third example evaluates the third degree polynomial $ax^3 + bx^2 + cx + d$.

For each example, we employed the methodology described in Section 5 to determine which of the faults within the controller were system-functionally redundant (SFR). As shown in Table 2, between 13% and 21% of the faults within the controller were SFR, meaning that they can not be detected by conventional means without somehow altering the controller-datapath pair. For these example circuits, remaining faults were system-functionally irredundant (SFI) and therefore can be caught with a test that exercises the controller-datapath pair as an indivisible unit. Our example circuits did not contain any controller-functionally redundant (CFR) faults; the synthesis method used for the finite state machine controllers did not allow redundancy.

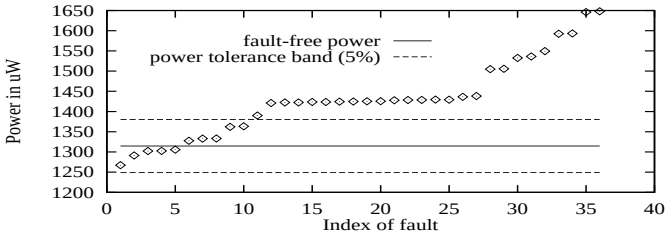
	Total Faults	SFR Faults	%Faults SFR
Diffeq	284	37	13.0%
Facet	177	36	20.3%
Poly	207	28	13.5%

Table 2: Breakdown of controller faults for the three examples.

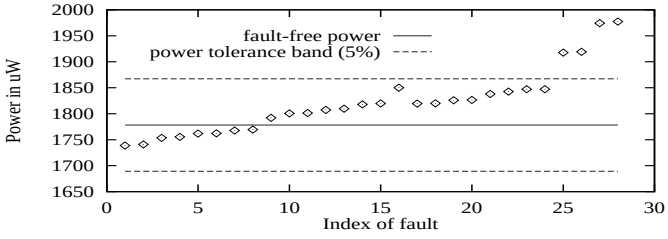
We next look at the power consumed by a datapath when driven by a controller that has an SFR fault. Figure 7(a) graphs results for the differential equation solver. The solid horizontal line shows the power consumed by the datapath when the controller is fault-free, 1679.35 uW. All power values shown are derived via Monte Carlo simulation, so that they represent power over a wide range of random input patterns. The two dashed lines show the threshold at which we detect a change in power; we have chosen the threshold to be 5%, so these lines lie at 1679.35 uW - 5% and 1679.35 uW + 5%. All possible SFR faults within the controller lie along the x-axis of the graph. The triangular data points show the power consumed by the datapath in the presence of the faults. The faults have no inherent order, and have been listed here with faults that affect multiplexer select lines only to the left of faults that affect register load lines. Within each group, the faults are sorted in order of increasing power. For this example, faults 1 through 19 affect only multiplexer select lines and faults 20 through 37 affect register load lines. The results for the differential equation solver show that all of the faults that affect only multiplexer select lines are too small to be detected, given a 5% power tolerance band. Of the nineteen faults that affect register load lines, fifteen can be detected.



(a) The differential equation solver.



(b) The facet example.



(c) The polynomial evaluator.

Figure 7: The effect of SFR faults within the controller on power consumption in a four-bit wide datapath for three examples.

	Monte Carlo power in uW	Test set 1 power in uW	Test set 2 power in uW	Test set 3 power in uW
fault-free	1679.35	1658.42	1603.92	1573.05
fault 1	1628.57 (-3.02%)	1613.09 (-2.73%)	1555.23 (-3.04%)	1533.07 (-2.54%)
fault 6	1680.35 (+0.06%)	1659.29 (+0.05%)	1604.44 (+0.03%)	1573.58 (+0.03%)
fault 21	1722.37 (+2.56%)	1714.02 (+3.35%)	1659.17 (+3.44%)	1629.84 (+3.61%)
fault 27	1833.40 (+9.17%)	1801.77 (+8.64%)	1749.36 (+9.07%)	1706.21 (+8.47%)
fault 37	2031.66 (+20.98%)	2001.37 (+20.68%)	1948.72 (+21.50%)	1953.14 (+24.16%)

(a) selected faults for the differential equation solver

	Monte Carlo power in uW	Test set 1 power in uW	Test set 2 power in uW	Test set 3 power in uW
fault-free	1778.33	1681.44	1655.54	935.48
fault 12	1807.29 (+1.63%)	1727.53 (+2.74%)	1699.09 (+2.63%)	961.29 (+2.76%)
fault 15	1819.77 (+2.33%)	1732.47 (+3.03%)	1707.37 (+3.13%)	960.07 (+2.63%)
fault 28	1974.30 (+11.02%)	1884.99 (+12.11%)	1862.05 (+12.47%)	1105.15 (+18.10%)

(b) selected faults for the polynomial evaluator

Table 3: Power in the presence of SFR faults for different test sets (percentage change from fault-free shown in parentheses).

Results for the facet example are shown in Figure 7(b). The facet example has a total of thirty-six SFR faults; six affect only multiplexer select lines, and thirty affect register load lines. From the graph, again we see that all six of faults that affect multiplexer selects are within the 5% power tolerance band, and so would not be detected. Twenty-six of the thirty register load line faults would be detected. The facet example has several sets of registers that load in parallel, and are driven by the same load line; this creates the potential for a single SFR fault to affect many registers, and therefore cause a large increase in power.

The polynomial example results are shown in Figure 7(c). Here, sixteen faults affect only multiplexer select lines. Again, none cause a big enough increase to move out of the 5% power tolerance band. Of the twelve faults that cause extra register loads, four are caught by the proposed method. The schedule for this example is such that many variables have relatively long lifespans. This translates into relatively small power effects for the SFR faults, because it is more likely that a given extra load will occur during a lifespan and be disruptive to the computation.

One important consideration mentioned in Section 5, is whether the effects of SFR faults on power are consistent for different short test sets. In order to test this hypothesis, we simulated the circuits for three different test sets, each with 1200 patterns. We generated the test sets by using different seeds in a TPGR, and we purposely choose a seed of almost all 0s for the third test set, with the idea that this test set would be less pseudorandom. Selected results for the differential equation solver and polynomial evaluator are shown in Table 3. Not all the faults are shown to save room. What we found is that while power varies for the different test sets, especially in the case of the polynomial evaluator and the third test set, the percentage increase over the fault-free case is reasonably consistent from test set to test set. This means that given a test set, one can simulate the circuit to find the fault-free power, and use that as a basis for the power-analysis based fault detection method.

7 Conclusions

This paper explores a method for detecting the class of system-functionally redundant faults in the controller of a controller-datapath pair. These faults do not affect the IO behavior of the controller-datapath pair, and as a result it is not possible to detect these faults with traditional methods unless the controller-datapath pair is broken apart during test with some kind of design-for-testability insertion. When the pair can not be separated for test, as is the case for embedded systems designed as hard cores, another approach is warranted. The proposed approach is built on the fact that the system-functionally redundant faults do change an important analog characteristic of the system: its power consumption. This work shows how a power analysis can be used to detect these faults without modifying the embedded core in any way.

References

[1] R. C. Aitken, "Finding Defects with Fault Models," *Proc. of the International Test Conference*, pp. 498–505, October 1995.

[2] P. Ashar and S. Devadas, "Optimum and Heuristic Algorithms for an Approach to Finite State Machine Decomposition," *IEEE Transactions on Computer-Aided Design*, pp. 296–310, March 1991.

[3] L. Benini and G. DeMicheli, "State Assignment for Low Power Dissipation," *Proc. of the IEEE Custom Integrated Circuits Conf.*, May 1994.

[4] L. Benini, P. Siegel and G. DeMicheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits," *IEEE Design and Test of Computers*, Dec. 1994.

[5] S. Bhatia and N. Jha, "Behavioral Synthesis for Hierarchical Testability of Controller/Datapath Circuits with Conditional Branches," *Proc. of the International Conf. on Computer Design*, June 1994.

[6] M. Breuer and J. Lien, "A Test and Maintenance Controller for a Module Containing Testable Chips," *Proc. of the International Test Conference*, October 1988.

[7] Compass Design Automation, "User Manuals for COMPASS VLSI V8R4.4," Compass Design Automation, Inc., 1993.

[8] S. Dey, V. Gangaram and M. Potkonjak, "A Controller-Based Design-for-Testability Technique for Controller-datapath Circuits," *Proc. of the International Conf. on Computer-Aided Design*, October 1995.

[9] S. Devadas and A. Newton, "Decomposition and Factorization of Sequential Finite State Machines," *IEEE Transactions on Computer-Aided Design*, pp. 1206–1217, November 1989.

[10] AT&T, "User Manuals for GENTEST_S 2.0," AT&T Bell Laboratories, 1993.

[11] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers: Boston, MA, 1992.

[12] S. Hellebrand and H. Wunderlich, "An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structure," *Proc. of the International Conf. on Computer-Aided Design*, June 1994.

[13] H. Harmanani, C. Papachristou, S. Chiu and M. Nourani, "SYNTEST: An Environment for System-Level Design for Test," *Proc. EURO-DAC 92*, Sept. 1992.

[14] R. Joersz and C. Kime, "A Distributed Hardware Approach to Built-In Self Test," *Proc. of the International Test Conference*, 1987. 1995.

[15] P. Landman and J. Rabaey, "Activity-Sensitive Architectural Power Analysis for the Control Path," *International Symposium on Low Power Design*, April 1995.

[16] M. Nourani, J. Carletta, and C. Papachristou, "A Scheme for Integrated Controller-Datapath Fault Testing," *Proc. of the Design Automation Conf.*, June 1997.

[17] C. Papachristou and J. Carletta, "Test Synthesis in the Behavioral Domain," *Proc. of the International Test Conference*, pp. 693–702, October 1995.

[18] VLSI Technology, "0.8-Micron CMOS VSC450 Portable Library," VLSI Technology, Inc., 1993.