Marcin IWIŃSKI, **Janusz SOSNOWSKI**
INSTYTUT INFORMATYKI, POLITECHNIKA WARSZAWSKA,
ul. Nowowiejska 15/19, 00-665 Warszawa

# Symulacja błędów przemijających w mikrokontrolerze satelitarnym

**Mgr inż. Marcin IWIŃSKI**

Ukończył studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej. Obecnie doktorant w Instytucie Informatyki na tym samym Wydziale. Entuzjasta nowych technologii, alternatywnych źródeł energii, systemów wbudowanych oraz ciekawych zastosowań mikrokontrolerów. Zawodowo pracuje jako programista oraz konstruktor urządzeń elektronicznych w firmie Albatross System sp. z o.o. specjalizującej się w produkcji profesjonalnych urządzeń alarmowych i monitoringu.

*e-mail: m.iwinski@ii.pw.edu.pl*

**Prof. dr hab. inż. Janusz SOSNOWSKI**

Ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej. Uzyskał tytuł profesora w 2006 r. Obecnie jest zatrudniony jako prof. zw. w Instytucie Informatyki na tym samym Wydziale. Jest autorem lub współautorem ok. 200 publikacji. Jego zainteresowania naukowe to wiarygodność systemów komputerowych (diagnostyka, tolerowanie błędów), architektura i interfejsy komputerów.

*e-mail: jss@ii.pw.edu.pl*

**Streszczenie**

W wielu zastosowaniach mikrokontrolerów należy brać pod uwagę wpływ na ich pracę błędów przemijających (zakłócenia elektromagnetyczne, promieniowanie kosmiczne itp.). Artykuł przedstawia metodę badania odporności na błędy przemijające w mikrokontrolerach. Bazuje ona na opracowanej platformie symulatora sprzężonego z obiektem badanym poprzez interfejs RS232C. Technika ta została zweryfikowana w badaniu mikrokontrolera przeznaczonego do sterowania zasilaniem pokładowym satelity. W artykule przedstawiono wyniki eksperymentów oraz wskazano możliwości programowego zwiększania odporności na błędy.

**Słowa kluczowe**: systemy wbudowane, mikrokontrolery, symulacja błędów, niezawodność.

## Transient fault simulation in a satellite microcontroller

**Abstract**

In many microcontroller applications the impact of transient faults (electromagnetic disturbances, cosmic radiation, etc.) on their operation has to be taken into account. The paper presents a new methodology of testing transient fault robustness in microcontrollers. It is based on the developed fault injection platform which is coupled to the tested object via RS 232C interface. A tested object (microcontroller) cooperates with real or modelled environment (partially controlled by a simulator). This technique has been successfully applied to testing a microcontroller used for managing the satellite on-board power subsystem (solar cells, batteries, power accumulation and distribution), Many transient fault simulation experiments have been performed and their results interpreted. In particular, there has been analysed the impact of faults on correct control flow of the program. Some simple fault detection and error recovery mechanisms have been included in the considerations. The presented methodology can be easily extended for other microcontrollers and communication interfaces. Time and code overheads are negligible so the simulation results are quite realistic.

**Keywords**: embedded systems, microcontrollers, fault simulation, reliability.

## 1. Wstęp

Układy mikrokontrolerów są powszechnie stosowane w różnych dziedzinach techniki w tzw. systemach wbudowanych charakteryzujących się tym, że realizują ściśle określone funkcje i działają w warunkach narażonych na różne zakłócenia np. elektromagnetyczne, promieniowanie kosmiczne (również to docierające do powierzchni ziemi), itp. Problemy te nabierają coraz większego znaczenia w miarę rozwoju technologii nanometrowych i obniżania poziomu zasilania (niższy poziom odporności na zakłócania). Stąd problem analizy efektów takich zakłóceń jak również metody ich tolerowania (np. na poziomie oprogramowania) są przedmiotem zainteresowania ośrodków naukowych jak również przemysłu [1-6]. W pracach tych istotną rolę odgrywają symulatory błędów, pozwalające analizować propagację ich efektów oraz skuteczność wprowadzanych mechanizmów detekcji i tolerowania.

Odczuwa się brak symulatorów błędów działających w ramach docelowej platformy mikrokontrolerów. Problem ten pojawił się przy projektowaniu systemu sterowania zasilaniem pokładowym satelity studenckiego zrealizowanym na bazie mikrokontrolera ATmega1280. Dla typowych orbit satelity wokół ziemi raportowano około 6 błędów przejmujących (SEU – single event upset) na dzień na 1 MB pamięci [7]. Stąd badanie wpływu błędów przemijających jest tu istotne [3]. Model opracowanego kontrolera był badany przy wykorzystaniu uniwersalnego symulatora FITS [4]. W celu umożliwienia przeprowadzenia badań na rzeczywistym systemie docelowym opracowano oryginalną metodę symulacji błędów poprzez dostępny interfejs RS 232C. Wykorzystuje ona system zarządzania eksperymentami (komputer IBM PC wraz z układem sprzężenia), rozszerzony protokół komunikacyjny o funkcje symulacji oraz opracowane proste procedury wstrzykiwania błędów zintegrowane z aplikacją docelową. Istotnym było uzyskanie zadowalającego poziomu sterowalności i obserwowalności eksperymentów symulacyjnych przy zminimalizowaniu instrumentacji programu badanego mikrokontrolera. System został zweryfikowany w badaniu wrażliwości na błędy przemijające w opracowanym mikrokontrolerze.

Koncepcję symulacji błędów przestawiono w rozdz. 2 i odniesiono ją do innych rozwiązań z literatury. Strukturę badanego mikrokontrolera oraz wyniki eksperymentów opisano w rozdz. 3. Możliwość wykorzystania zdobytego doświadczenia w badaniu innych mikrokontrolerów przedstawiono w rozdz. 5.

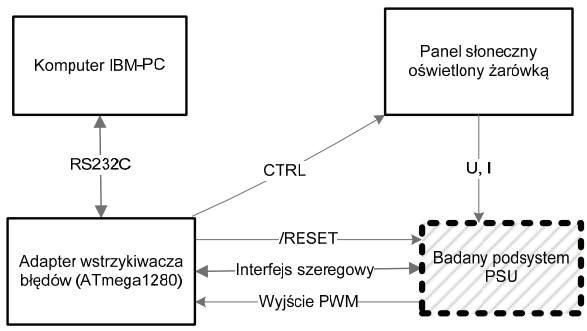
## 2. Platforma symulacji błędów

Symulatory błędów stanowią bardzo przydatne narzędzie do badania wrażliwości na błędy systemów komputerowych. Wyróżnić tu można dwie klasy symulatorów: działające na modelu (programowym) systemu docelowego oraz działające na rzeczywistym systemie (system docelowy) [1, 2, 3]. Te pierwsze zwykle mają dobrą sterowalność i obserwowalność eksperymentów, ale obarczone są problemem dokładności modelu. Symulatory drugiej kategorii są uzależnione od platformy sprzętowo programowej systemu badanego oraz środowiska jego pracy. Jest to szczególnie kłopotliwe w przypadku systemów wbudowanych, zwykle bazujących na specyficznych platformach, dla których nie ma dostępnych symulatorów. Ten problem napotkali autorzy w badaniach sterownika przeznaczonego dla zastosowań kosmicznych (sterowanie zasilaniem dla wielu podsystemów pokładowych satelity).

Pewnym rozwiązaniem ww. problemu jest koncepcja wbudowania jądra symulatora błędów w system docelowy (poprzez pewną minimalną instrumentację jego oprogramowania) oraz zarządzania eksperymentami poprzez zewnętrzny komputer sprzężony z systemem docelowym i jego środowiskiem poprzez odpowiedni adapter (zapewnia sterowanie i monitorowanie). Implementacja tej koncepcji zależy od konkretnego systemu docelowego. Niemniej jednak w większości takich systemów mamy albo dostępne wolne niewykorzystywane interfejsy albo też można

w tym celu wykorzystać dostępne interfejsy przez rozszerzenie ich protokołu o funkcje symulatora. Jądro symulatora najprościej zrealizować w oparciu o system przerwań (obsługa dodatkowych funkcji symulacji). Narzut programowy i czasowy jest tu niewielki i zwykle akceptowalny dla większości systemów. Istotnym jest również zapewnienie możliwości wpływania na środowisko zewnętrze badanego systemu oraz obserwacji zachowania się badanego systemu (np. monitorowanie i analiza zewnętrznych sygnałów wyjściowych). Poniżej przedstawiono rozwiązanie tych problemów w przypadku sterownika satelitarnego.

Platforma do symulacji błędów składa się z komputera zarządzającego eksperymentem (IBM-PC), mikrokontrolera pełniącego rolę adaptera, panelu słonecznego z oświetlaniem (żarówka) oraz testowanego obiektu – sterownika PSU (rys. 1).




Rys. 1. Diagram platformy do symulacji błędów
Fig. 1. Layout of fault injection platform

Komputer pełni rolę terminala do konfigurowania eksperymentów, odbierania i obróbki danych przesyłanych przez adapter (mikrokontroler) symulatora błędów. Komunikacja z adapterem odbywa się poprzez interfejs RS232. Odbierane dane są już w odpowiedni sposób sformatowane, tak, aby było możliwe ich bezpośrednie eksportowanie do arkusza kalkulacyjnego (np. Excel) i dalsze przetwarzanie.

Mikrokontroler symulujący błędy (adapter) spełnia w całej platformie najważniejszą rolę. Generuje on rozkazy symulacji błędów i przekazuje je do badanego podsystemu PSU poprzez interfejs szeregowy, wykorzystujący układy UART. Badany mikrokontroler satelitarny ma wbudowane 3 układy UART, z czego jeden nie jest wykorzystywany przez badany program i został on użyty tylko do komunikacji z symulatorem błędów. Mikrokontroler adaptera kontroluje (sterowanie i monitorowanie) działania badanego podsystemu. Ma on możliwość sprzętowego inicjalizowania (resetowania) podsystemu PSU poprzez wyprowadzenie /RESET. Dodatkowo może też sprawdzać poprawność działania kluczowego elementu programu PSU, czyli algorytmu MPPT-P&O [8] sterującego mocą odbieraną z paneli słonecznych. Odbywa się to poprzez sygnał CNTR - włączanie klucza tranzystorowego P-MOS, który spowoduje podanie na jedno z wejść mocy podsystemu PSU zasilania z panelu słonecznego. Mikrokontroler może wtedy sprawdzić, czy uzyskana wartość sterowania PWM (długość impulsów) jest zbliżona do tej, która została uzyskana przy wcześniejszym uruchomieniu wzorcowym. Sygnał PWM został podpięty do wejścia licznikowego mikrokontrolera symulatora. Panel słoneczny jest przez cały czas eksperymentu oświetlany jednakowym światłem z żarówki. Wymagane jest włączenie oświetlenia panelu kilkanaście minut przed przeprowadzaniem eksperymentu, aby ustabilizować temperaturę panelu na stałym poziomie. Model otoczenia może być bardziej rozbudowany (np. sterowanie zmianą oświetlenia, dodawanie obciążenia prądowego).

Adapter symulatora wysyła do badanego systemu rozkazy symulacji błędów zgodnie ze zdefiniowanym protokołem komunikacji z badanym systemem (PSU). Przykładowo przekłamanie licznika instrukcji (Program Counter) jest specyfikowane poprzez wysłanie do PSU sekwencji 3 bajtów: pierwszy bajt to specjalny znacznik 0xFB, kolejne 2 bajty stanowią adres, pod który ma zostać wykonany skok symulujący przekłamanie. Podobnie (używając innego znacznika rozkazowego) można specyfikować inne błędy np. zaburzenie stanu komórki pamięci RAM lub rejestrów procesora (z wyjątkiem rejestru R15). Rozkazy symulacji błędu generują w systemie badanym (PSU) przerwanie, którego obsługa realizuje żądane zakłócenie stanu.

Odpowiedni scenariusz eksperymentu (liczba błędów, rozkład w czasie i przestrzeni, itp.) jest programowany przez użytkownika i następnie realizowany przez adapter generujący odpowiednią sekwencję rozkazów. Eksperyment zwykle jest złożony z wielu testów. Jeden test odpowiada symulacji jednego błędu. System umożliwia realizację dwu strategii eksperymentów, sekwencja pojedynczych testów lub grup testów. W pierwszym przypadku przed każdym testem system jest inicjalizowany, po wysłaniu rozkazu symulacji błędu jest monitorowany stan systemu przez określny czas (typowo 9 min), po zarejestrowaniu wyniku testu realizowany jest kolejny test sekwencji. W drugim scenariuszu po inicjalizacji generowana jest sekwencja rozkazów symulacji np. 10 i dopiero po jej zakończeniu badany jest stan systemu a po zarejestrowaniu wyniku przechodzi się do kolejnej sekwencji.

Należy zaznaczyć, że obsługa symulacji błędów wymagała pewnych modyfikacji (instrumentacji) oryginalnego programu sterującego. Narzut programowy ograniczył się do zajęcia na stałe jednego rejestru R15, który i tak jest używany przez kompilator tylko przy bardzo dużej liczbie zmiennych lokalnych w programie. W testowanym programie nie był on wykorzystywany. Procedura obsługi przerwania odbieranych rozkazów symulacji błędów zawarła się w kilkunastu instrukcjach asemblerowych (pomijalny narzut kodu i czasu). Rozkazy te mogą być odbierane tylko przy odblokowanych przerwaniach. Okresy takiej blokady były bardzo krótkie. Stąd uzyskanie odpowiedniego rozkładu w czasie symulacji błędów nie było problemem.

## 3. Wyniki eksperymentów

Program sterujący podsystemu PSU jest przechowywany w pamięci FLASH, wykorzystuje on małą liczbę komórek pamięci RAM oraz realizuje pewien algorytm iteracyjny. Stąd najbardziej krytyczne są błędy przemijające zakłócające przepływ sterowania programu. Problem ten stał się przedmiotem opisanych niżej badań eksperymentalnych, w których błędy były symulowane przez zmianę stanu licznika rozkazów.

Wyniki badań przedstawiono w tab. 1 i 2. Odpowiadają one następującemu zaprogramowanemu scenariuszowi eksperymentów). Po zainicjalizowaniu modułu PSU generowanie jest 100 serii po 10 wstrzyknięć błędów. Adresy skoków są generowane losowo w mikrokontrolerze wstrzykującym z zakresu 0 – 9kB, bo tyle faktycznie zajmuje program sterujący PSU. Próba skoku poza zakres programu spowoduje w efekcie reset mikrokontrolera. Przed każdą serią 10 kolejnych wstrzyknięć PSU jest resetowany, oraz odczytywane są statystyki resetów zbierane przez program sterujący tym podsystemem. Widać, zatem, że efekty każdych dziesięciu kolejnych błędów kumulują się. Przed każdym wstrzyknięciem błędu generowane jest pewne opóźnienie z zakresu od 0,5 do 9s (z rozdzielczością do 5 mikrosekund), aby losowo rozłożyć w czasie pojawianie się kolejnych przekłamań. Po każdym błędzie symulator odczekuje dodatkowo 2 sekundy, aby dać możliwość zadziałania wbudowanym mechanizmom obsługi błędów w PSU lub modułowi watchdog. Następnie weryfikowane jest działanie algorytmu MPPT. Odbywa się ono poprzez włączenie klucza tranzystorowego, oraz po kolejnej sekundzie kontroli wartości PWM. System porównuje ją ze wzorcem uzyskanym w podobny sposób dla przebiegu bez błędów.

Tab. 1 i 2 przedstawiają wyniki eksperymentów dla dwu wersji programów: A – wersja podstawowa oraz B – wersja zmodyfikowana. W wersji podstawowej błędy są wykrywane przez watchdog (brak jego inicjalizacji programowej w odpowiednim czasie) oraz inne mechanizmy sprzętowe (detekcja niepoprawnego kodu instrukcji, pobranie kodu spoza obszaru programu). Wszyst-

kie te przypadki powodują zerowanie (reset) mikrokontrolera. W wersji zmodyfikowanej dodano programową weryfikacje poprawności przepływu sterowania. Jest to znacznie uproszczona wersja mechanizmu CFCSS opisanego w [9]. Badane są sygnatury (suma modulo 2) kilkunastu punktów kontrolnych programu sterującego. Dzięki temu narzut kodu programu jest poniżej 1% (oryginalny algorytm CFCSS wymaga kilkudziesięciu procent, bo sygnatury CRC są dla każdego bloku programu). Wykryty błąd sygnatury wymusza zerowanie mikrokontrolera. Każde zerowanie kontrolera powoduje powrót do kontynuacji realizacji programu sterującego. Wynikające stąd chwilowe zaburzenia pracy praktycznie nie wpływają na sterowanie zasilaniem. Tabela 1 pokazuje rozkład zdarzeń zerowania systemu oraz niewykrytych błędów (krytyczne – nieakceptowalne oraz anomalie, czyli pewne zaburzenia wpływające na wydajność sterowania). Wyniki obejmują eksperyment z symulacją 1000 błędów (100 sekwencji błędów po 10). Dla wersji B pogrubionym drukiem zaznaczono liczbę błędów (478) wykrytych przez dodany mechanizm detekcji błędów. Dla wersji B uzyskano zdecydowanie mniej sytuacji z błędami krytycznymi jak również znacznie mniej anomalii.

Tab. 1. Wyniki eksperymentu dla wersji A i B programu sterującego
Tab. 1. Experiment results for A and B versions of the control program

| Wersja | Źródło zerowania (reset) | | Niewykryte błędy | |
|---|---|---|---|---|
| | Watchdog | Inne | anomalie | krytyczne |
| A | 492 | 22 | 486 | 25 |
| B | 472 | 11 + **478** | 39 | 4 |

Tab. 2. Rozkład sygnałów zerowania systemu dla wersji B
Tab. 2. Distribution of reset signals in the system – version B

| Ilość wystąpień | | Resety Watchdog | | Resety od CFCSS | | Inne resety | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 2 | 8 | 7 | 0 | 0 |
| 5 | 5 | 3 | 3 | 7 | 6 | 0 | 0 |
| 6 | 11 | 4 | 4 | 6 | 5 | 0 | 0 |
| 20 | 14 | 5 | 5 | 5 | 4 | 0 | 0 |
| 10 | 8 | 6 | 6 | 4 | 3 | 0 | 0 |
| 7 | 4 | 7 | 4 | 3 | 5 | 0 | 1 |
| 5 | 2 | 3 | 5 | 6 | 4 | 1 | 1 |

W tab. 2 przedstawiono rozkład liczby błędów generujących sygnał zerowania (reset). Każdy wiersz tabeli odpowiada 2 grupom sekwencji symulowanych błędów (10 błędów w sekwencji) z takim samym rozkładem przyczyn zerowania (kolumny reset). Grupy te rozróżniono przez zacienienie. Uzyskano 14 różnych rozkładów (7 w kolumnach jasnych i 7 w zacienionych). W pierwszej parze kolumn podano liczbę występujących sekwencji o danym rozkładzie (suma tych liczb to 100 sekwencji testowych). Różnorodność tych wzorów potwierdza, że poszczególne mechanizmy detekcji błędów są aktywowane w różnym zakresie dla różnych serii testów, co wskazuje na ich pewną ortogonalność, czyli komplementarne własności detekcyjne.

## 4. Wnioski

Przedstawiona metodyka badania odporności (lub wrażliwości) mikrokontrolera na błędy przemijające potwierdziła jej praktyczną przydatność. Istotna zaletą przedstawionego rozwiązania jest prosta implementacja środowiska symulacyjnego. Interakcja z badanym obiektem wymaga odpowiedniej adaptacji jego oprogramowania sterującego. Dotyczy to tylko jednego z zewnętrznych interfejsów i obsługi dodatkowego przerwania. Przeprowadzone eksperymenty ograniczono do symulacji błędów zaburzających przepływ sterowania programu. Można to łatwo rozszerzyć na błędy zakłócania stanu pamięci programu, danych i rejestrów procesora. W porównaniu z klasycznymi symulatorami błędów [1, 2] metoda zapewnia możliwość badań na rzeczywistych systemach i praktyczne na dowolnej docelowej platformie. Klasyczne symulatory uniwersalne zwykle maja tu istotne ograniczenia ponadto wymagają dużych zasobów komputerowych (pamięć kodu) niedostępnych w systemach wbudowanych.

Pewnym problemem jest zapewnienie sprzężenia badanego obiektu ze środowiskiem. W celu zapewnienia dobrej sterowalności wygodnie jest dysponować pewnym modelem takiego środowiska. W przypadku środowiska o dużym stopniu interakcji znalezienie takiego modelu jest wyzwaniem samym w sobie (problem ten był przedmiotem innych prac w Instytucie [10, 11]).

Przedstawione rozwiązanie wymaga odpowiedniej adaptacji do badanego mikrokontrolera (znalezienie wolnego interfejsu lub rozszerzenie dostępnego, instrumentacja programu aplikacji).Ten wymóg można uwzględnić w wymaganiach projektowych systemu. Warto zwrócić uwagę, że dużym ułatwieniem w realizacji eksperymentów jest dysponowanie wbudowanym sprzętowym mechanizmami symulacji błędów (np. dostępne w standardzie interfejsu uruchomieniowego IEEE Nexus [12]). Znika wtedy problem instrumentacji aplikacji programowej ponadto mamy tu pełną swobodę w wyborze momentów symulacji błędów (duża sterowalność eksperymentów). Problem oceny wyników testów jest uzależniony od aplikacji i zwykle wymaga monitorowania sygnałów zewnętrznych. Możliwe jest zwiększenie obserwowalności przez dodatkowe przekazywanie z badanego systemu informacji o wybranych jego stanach.

## 5. Literatura


[1] Arla J. i inni: Comparison of physical and software-implemented fault injection techniques, IEEE Trans. on Computers, vol.52, no.9, 2003, 1115-1133.

[2] Benso A., Prinetto P.: Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation. Kluwer Academic Publishers, 2003.

[3] Campagna S., Violante M.: A framework to support the design of COTS based reliable space computers for on-board data handling, Proc. of IEEE IOLTS Symposium, 2010, pp.91-96.

[4] Gawkowski P., Sosnowski J.: Experiences with software implemented fault injection, Proc. of the International Conference on Architecture of Computing Systems, VDE Verlag GMBH, 2007, 73-80.

[5] Rebaudengo M., Reorda M., Villante M.: A new software based technique for low cost fault tolerant application, Proc. of IEEE Annual Reliability and Maintainability Symposium, 2003, 23-28.

[6] Skarin D., Karlsson J.: Software implemented detection and recovery of soft errors in a break by wire system, Proc. of 7th European Dependable Computing Conference, IEEE Comp. Soc., 2008, 145-154.

[7] Lovellette M. N. i inni: Strategies for Fault-Tolerant, Space-Based Computing: Lessons Learned from the ARGOS Testbed, Aerospace Conference Proceedings, 5-2109- 5-2119 vol.5, 2002.

[8] Sera D. i inni: Improved MPPT algorithms for rapidly changing environmental conditions, Aalborg University/Institute of Energy Technology, Aalborg, Denmark.

[9] Oh N., Shirvani P.P., McCluskey E.J.: Control-flow checking by software signatures, IEEE Transactions on Reliability, vol. 51, no. 1, pp. 111–122, 2002.

[10] Trawczynski D., Sosnowski J., Gawkowski P.: Analyzing fault susceptibility of ABS microcontroller, Proc. International Conference on Computer Safety, Reliability, and Security SAFECOMP'08, Newcastle, U.K, 2008, 320-372.

[11] Gawkowski P. et al.: Testing Fault Robustness of Model Predictive Control Algorithms. H. Giese (Ed.): ISARCS 2010, LNCS 6150, pp. 109–124, Springer-Verlag Berlin Heidelberg 2010.

[12] Fidalgo A. V., Alves G. R., Ferreira J. M.: Real Time Fault Injection Using Enhanced OCD – A Performance Analysis, Proceedings of the 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06), 2006.


*otrzymano / received: 13.05.2011*
*przyjęto do druku / accepted: 04.07.2011* *artykuł recenzowany*