А.В. Ворон

Метод аппаратной детекции и локализации одиночных сбоев на основе инварианта разности весов дуальных кодов в системах SBNS и NFNS

Аннотация. В статье представлен метод аппаратного обнаружения, локализации и автоматического исправления одиночных радиационных сбоев (SEU) в арифметико-логических устройствах и кэш-памяти, функционирующих в масштабированной системе счисления Бергмана (SBNS) и новой фибоначчиевой системе счисления (NFNS). В качестве альтернативы классическим кодам Хэмминга предлагается контроль целостности данных на основе инварианта разности весов Хэмминга (PopCount) для дуальной пары разрешенных кодовых состояний – рабочего кода «А» и контрольного кода «Б». Показано, что отклонение дельты весов от эталонного значения позволяет не только зафиксировать факт сбоя без вычисления громоздких синдромов, но и мгновенно локализовать поврежденный код и физическую природу ошибки (инверсию единицы в ноль или нуля в единицу). Рассмотрен краевой эффект симметрии для числа 1 в системе NFNS и предложено микроархитектурное решение в виде логического предохранителя.

Ключевые слова: масштабированная система счисления Бергмана (SBNS); новая фибоначчиева система счисления (NFNS); дуальный базис; инвариант разности весов; PopCount; радиационные сбои (SEU); самоисцеляющаяся память (SRAM); фильтр четности.

Введение. Современная бортовая и космическая микроэлектроника функционирует в условиях жесткого воздействия ионизирующего излучения и тяжелых заряженных частиц. Попадание заряженной частицы в активную область полупроводникового кристалла вызывает одиночные сбои (спонтанную инверсию логического состояния триггеров регистров или ячеек статической оперативной памяти (SRAM)) [8, 9]. При прохождении частицы под острым углом к поверхности кристалла возникает каскад вторичных ионизаций, приводящий к множественным кластерным сбоям [9].

Традиционные методы парирования радиационных сбоев в архитектуре фон Неймана опираются на два ресурсоемких подхода [8, 9].

1. Программная и аппаратная избыточность кодирования (коды Хэмминга, ECC, CRC) [6, 8]. Формирование синдромов ошибок для широких шин данных и строк кэш-памяти требует построения глубоких комбинационных деревьев из логических элементов «исключающее ИЛИ» (XOR). Это увеличивает задержки распространения сигналов, снижает предельную тактовую частоту и требует выделения дополнительной площади памяти под контрольные биты [6, 8].

2. Трехкратное дублирование вычислительных ядер с голосованием «два из трех» увеличивает энергопотребление, тепловыделение и массу бортовой аппаратуры [7, 9].

Опираясь на фундаментальную теорию иррациональных оснований и систем счисления Фибоначчи [1–5], автор ранее формализовал новую систему счисления Фибоначчи (New Fibonacci Number System, NFNS) [10, 11] и масштабированную систему счисления Бергмана (Scaled Bergman Number System, SBNS) с коэффициентом масштабирования $k = 0{,}2$ [12]. Применение аппаратного фильтра четности («правила четности») удерживает любое число внутри процессора строго в рамках пары детерминированных разрешенных состояний: развернутого рабочего кода «А» и свернутого контрольного кода «Б».

Целью настоящей работы является разработка и микроархитектурное обоснование метода сквозной аппаратной детекции, локализации и мгновенного самоисцеления одиночных сбоев в арифметико-логическом устройстве (АЛУ) и кэш-памяти процессора на основе инварианта разности весов дуальных кодов.

Основная часть.

Математический принцип инварианта разности весов. В системах счисления SBNS [1, 12] и NFNS [2–5, 10, 11] для любого корректно представленного числа выполняется фундаментальное условие целостности: разность весов Хэмминга (количества активных единичных разрядов, вычисляемых функцией PopCount) контрольного кода «Б» и рабочего кода «А» является константой и равна единице

$$\Delta = \text{PopCount}(\text{Код Б}) - \text{PopCount}(\text{Код А}) = 1.$$

Аппаратно подсчет количества единиц реализуется на базе компактных комбинационных каскадов сумматоров (деревьев Уоллеса или инверсных транзисторных цепочек), что требует в разы меньше вентилей и логических уровней, чем декодирование матриц Хэмминга [6, 8]. При этом на шине данных обеспечивается нулевая битовая избыточность, поскольку контрольный код «Б» в АЛУ формируется и проверяется параллельно на сопряженных логических вентилях [11, 12].

При возникновении радиационного сбоя (SEU), инвертирующего один бит в любой из кодовых последовательностей, инвариант $\Delta = 1$ мгновенно нарушается. В отличие от классического побитового компаратора (XOR), который при несовпадении кодов лишь фиксирует факт ошибки («данные невалидны»), но не способен указать, какой именно из двух кодов поврежден, значение дельты задает прямой вектор локализации и исправления:

$\Delta = 1$ (Штатный режим). Данные валидны, сбои отсутствуют.

$\Delta = 2$ (Угасание бита в Коде А). В рабочем коде произошла инверсия $(1 \to 0)$. Контрольный код «Б» не затронут и принимается за математический эталон.

$\Delta = 0$ (Возгорание бита в Коде А). В рабочем коде под воздействием наводки появилась ложная единица $(0 \to 1)$. Код «Б» принимается за эталон.

$\Delta = -1$ (Возгорание бита в Коде Б). В контрольном коде произошла инверсия $(0 \to 1)$. Рабочий код «А» не поврежден и выступает эталоном.

$\Delta = 3$ (Угасание бита в Коде Б). В контрольном коде потеряна единица $(1 \to 0)$. Код «А» выступает эталоном.

$\Delta \notin \{-1, 0, 1, 2, 3\}$. Зафиксировано критическое разрушение дуального базиса вследствие каскадного многократного сбоя; генерируется аппаратное прерывание (HARDWARE EXCEPTION).

В ПРИЛОЖЕНИИ А приведена программно-логическая модель диагностического модуля АЛУ, реализующая автоматический перехват и восстановление поврежденного регистра без остановки вычислительного конвейера. При этом приведена поведенческая (функциональная) модель проверки.

Побитовое сравнение кодов «А» и «Б» через стандартные вентили XOR в системе SBNS [1, 12] невозможно напрямую, так как эти коды представляют собой разные битовые комбинации (развернутую и свернутую формы). Предварительная свертка одного кода к виду другого потребовала бы прогона данных через дополнительные каскады сумматоров прямо в контуре контроля. Метод вычитания весов (PopCount) обходит это ограничение: он оперирует не позициями разрядов, а структурным весом кода, совмещая детекцию и локализацию сбоя в одном такте вычитания.

Краевой эффект в системе NFNS и его устранение. При масштабировании описанного алгоритма на целочисленную Новую фибоначчиеву систему счисления (NFNS) [2–5, 10, 11] выявляется математическое исключение, связанное с представлением числа 1.

В системе SBNS [12] число 1 разводится по разным диапазонам разрядной сетки: рабочий код «А» занимает разряды $\{3, -2, -3, -4\}$ $(\text{PopCount} = 4)$, а контрольный код «Б» разворачивается в разряды $\{2, 1, -2, -3, -4\}$ $(\text{PopCount} = 5)$, что обеспечивает $\Delta = 5 - 4 = 1$. Но в базисе NFNS, где младшие опорные веса задаются числами Фибоначчи $F_1 = 1$ и $F_2 = 1$, дуальные формы единицы оказываются полностью симметричными.

Рабочий код «А»: активирован разряд F_1 (битовая маска 01), следовательно, $\mathrm{PopCount}(\text{Код А}) = 1$.

Контрольный код «Б»: активирован дуальный разряд F_2 (битовая маска 10), следовательно, $\mathrm{PopCount}(\text{Код Б}) = 1$.

Подстановка этих значений в формулу инварианта дает

$$\Delta = \mathrm{PopCount}(\text{Код Б}) - \mathrm{PopCount}(\text{Код А}) = 1 - 1 = 0.$$

В исправном состоянии при обработке единицы в режиме NFNS диагностический модуль зафиксирует $\Delta = 0$, ошибочно классифицирует ситуацию как «возгорание ложного бита в Коде А», обнулит легитимный разряд F_1 и заблокирует корректное вычисление.

Для устранения этого краевого эффекта симметрии в начало комбинационной логики диагностического модуля вводится аппаратный краевой предохранитель (ПРИЛОЖЕНИЕ Б).

На уровне кремниевой топологии такая проверка реализуется всего несколькими логическими вентилями, активируемыми флагом целочисленного режима REG_MODE == 0, и полностью исключает ложные срабатывания системы защиты.

Представление машинного нуля и защита от состояния обрыва шины. В базовой теории систем SBNS и NFNS [10–12] кодовые таблицы строятся для множества натуральных чисел, однако в реальной микроархитектуре АЛУ и кэш-памяти (SRAM) необходимо корректно обрабатывать состояние логического нуля (например, после аппаратного сброса регистров). Использование полностью нулевых кодовых векторов ($\mathrm{PopCount}(\text{Код А}) = 0$, $\mathrm{PopCount}(\text{Код Б}) = 0$) недопустимо по двум причинам: во-первых, это нарушает базовый инвариант $(\Delta = 0 - 0 = 0)$, вызывая ложное срабатывание защиты; во-вторых, в радиационно-стойкой электронике полностью нулевая комбинация должна трактоваться как аварийный обрыв питания или шины данных.

Для устранения этой уязвимости вводится ненулевой базис машинного нуля. Число 0 кодируется нулевым значением информационных разрядов рабочего кода «А» $(\mathrm{PopCount}(\text{Код А}) = 0)$ при активации одного выделенного служебного бита нуля $(Z = 1)$ в контрольном коде «Б» $(\mathrm{PopCount}(\text{Код Б}) = 1)$. Данное решение обеспечивает выполнение единого инварианта $(\Delta = 1 - 0 = 1)$ для нуля без расширения краевого предохранителя и гарантирует однотактную коррекцию одиночного сбоя: возгорание любого информационного бита в коде «А» дает $\Delta = 0$ (восстанавливается по эталонному коду «Б» с флагом $Z = 1$), а угасание бита Z в коде «Б» приводит к полному обнулению обеих половин пары $(\Delta = 0)$, детектируемому как аппаратный сбой шины BUS_ZERO_FAULT.

Микроархитектура самовосстанавливающейся кэш-памяти (SRAM). В радиационно-стойких процессорах статическая кэш-память (SRAM) занимает до 80% площади кристалла и является главной мишенью для тяжелых заряженных частиц [8, 9]. Перенос логики контроля по инварианту разности весов (PopCount) на уровень кэш-памяти при дуальном хранении строк (парами «Код А + Код Б») обеспечивает ряд преимуществ перед классическими кодами коррекции ошибок Хэмминга (SEC-DED) [6, 8].

Классическая ECC-память при чтении поврежденной строки исправляет бит на лету в выходном буфере, но в самой ячейке памяти сбойное состояние сохраняется до следующей перезаписи, создавая риск накопления двойной неисправляемой ошибки [6, 9]. В дуальной архитектуре отклонение дельты $\Delta = 2$ или $\Delta = -1$ однозначно указывает, какая именно из половин пары повреждена. Контроллер кэша немедленно формирует цикл обратной записи (Write-Back) восстановленного эталонного значения прямо в процессе чтения, физически устраняя последствия радиационного удара.

В отличие от АЛУ, где код «Б» генерируется комбинационно «на лету», в SRAM необходимо физически хранить обе кодовые последовательности. Поскольку рабочий и контрольный коды имеют различную битовую структуру, это требует удвоения массива ячеек памяти. Для сохранения экономической и энергетической эффективности кристалла применяется селективное (зонное) дублирование [8, 9]. Основной массив кэша ($L2$/$L3$)

защищается компактными контрольными суммами или классическим кодом Хэмминга [6]. Критически важная память (кэш первого уровня $L1$, массивы тегов, таблица векторов прерываний, управляющие регистры REG_MODE) строится на базе дуального хранения с контролем по инварианту $\Delta = 1$. В этой зоне удвоение площади оправдано гарантией мгновенного самовосстановления без потери тактов конвейера.

Архитектура зеркальных банков памяти. При физической компоновке дуальной кэш-памяти на кристалле размещение рабочего кода «А» и контрольного кода «Б» в одной физической строке SRAM недопустимо. Тяжелая заряженная частица, проходящая через кремний под углом, вызывает кластерный сбой (MBU), поражающий группу соседних ячеек [9]. Если коды «А» и «Б» расположены рядом, один радиационный трек способен одновременно повредить оба кода, разрушив инвариант дельты и лишив АЛУ неповрежденного эталона.

Для нейтрализации угроз MBU предлагается архитектура зеркальных банков памяти. Основной банк хранит рабочие коды «А». Зеркальный банк хранит контрольные коды «Б» и физически разнесен на кристалле на расстояние, превышающее радиус вторичной ионизации трека частицы [9]. Такое разделение обеспечивает параллельное считывание обеих половин дуальной пары по независимым шинам за один такт: данные поступают в аппаратный модуль (ПРИЛОЖЕНИЕ А) одновременно, не создавая задержек на мультиплексирование длинных строк.

Заключение.

1. Предложенный метод детекции на основе инварианта разности весов дуальных кодов ($\Delta = \mathrm{PopCount}(\text{кодБ}) - \mathrm{PopCount}(\text{кодА}) = 1$) позволяет отказаться от сложных логических деревьев вычисления синдромов Хэмминга в системах счисления SBNS и NFNS. Значение дельты обеспечивает однотактную локализацию поврежденного кода и типа инверсии бита, позволяя АЛУ выполнять самоисцеление регистров без остановки конвейера.

2. Выявлен краевой эффект симметрии при представлении числа 1 в целочисленной системе NFNS ($\Delta = 0$ в штатном режиме) и предложена его аппаратная компенсация в виде микроархитектурного логического предохранителя.

3. Обоснована архитектура зеркальных физически симметричных банков кэш-памяти с селективным дублированием критических зон. Данное решение защищает вычислительную среду от кластерных радиационных сбоев (MBU) и обеспечивает фоновое восстановление поврежденных ячеек SRAM в цикле чтения.

Список литературы

1. Бергман Дж. Система счисления с иррациональным основанием // Mathematics Magazine. – 1957. – Vol. 31, № 2. – P. 98–110.

2. Стахов А.П. Введение в алгоритмическую теорию измерения. – М.: Советское радио, 1977. – 288 с.

3. Стахов А.П. Коды золотой пропорции. – М.: Радио и связь, 1984. – 152 с.

4. Стахов А.П. Математика гармонии. От Евклида к современной математике и компьютерным наукам. – Сингапур: World Scientific, 2009. – 748 с.

5. Цекендорф Э. Представление натуральных чисел в виде суммы чисел Фибоначчи или чисел Люка // Bulletin de la Société Royale des Sciences de Liège. – 1972. – Vol. 41. – P. 179–182.

6. Хэмминг Р.У. Коды с обнаружением и исправлением ошибок // The Bell System Technical Journal. – 1950. – Vol. 29, № 2. – P. 147–160.

7. Лайонс Р.Э., Вандеркалк У. Применение тройного модульного резервирования для повышения надежности ЭВМ // IBM Journal of Research and Development. – 1962. – Vol. 6, № 2. – P. 200–209.

8. Пархами Б. Компьютерная арифметика: сложность и проектирование отказоустойчивых систем. – Нью-Йорк: Oxford University Press, 2010. – 496 с.

9. Корен И., Кришна К.М. Отказоустойчивые системы. – 2-е изд. – Кембридж: Morgan Kaufmann, 2020. – 422 с.

10. Ворон А.В. Избыточность кода Фибоначчи и разработка новой системы счисления Фибоначчи – NFNS // «Академия Тринитаризма», М., Эл № 77-6567, публ. 30093, 25.07.2026. – URL: https://www.trinitas.ru/rus/doc/0016/001k/00166136.htm

11. Ворон А.В. Разработка избыточной системы счисления Фибоначчи NFNS с дуальным базисом для помехоустойчивых вычислительных сред // «Академия Тринитаризма», М., Эл № 77-6567, публ. 30171, 21.09.2026. – URL: https://www.trinitas.ru/rus/doc/0016/001k/00166174.htm

12. Ворон А.В. Масштабированная система счисления Бергмана (SBNS) и фильтр Ворона как новый базис вычислительной техники // «Академия Тринитаризма», М., Эл № 77-6567, публ. 30210, 22.10.2026. – URL: https://www.trinitas.ru/rus/doc/0023/001a/00231195.htm

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

```
def voron_hardware_diagnostic_and_fix(
    code_a: dict, code_b: dict, processor_instance
) -> tuple:
    """
    Поведенческая модель модуля перехвата, локализации и
    исправления одиночных сбоев (SEU) по инварианту весов.
    Поддерживает ненулевой базис машинного нуля (ключ 'Z').
    """
    # 0. Проверка на критический обрыв шины / обесточивание
    is_all_zero_a = all(v == 0 for v in code_a.values())
    is_all_zero_b = all(v == 0 for v in code_b.values())
    if is_all_zero_a and is_all_zero_b:
        raise RuntimeError(
            "HARDWARE EXCEPTION: Нулевой вектор на обеих шинах "
            "(обрыв питания или угасание Z-бита нуля)."
        )

    # 1. Штатный режим машинного нуля (Zero-Offset: Z = 1)
    is_zero_b = (
        code_b.get("Z") == 1
        and all(v == 0 for k, v in code_b.items() if k != "Z")
    )
    if is_all_zero_a and is_zero_b:
        return (
            code_a,
            code_b,
            "STATUS 200 (ZERO-OFFSET): Машинный ноль валиден. "
            "Инвариант равен 1.",
        )

    # 2. Краевой предохранитель для числа 1 в режиме NFNS
    if nfns_edge_guard(
        code_a, code_b, processor_instance.REG_MODE
    ):
        return (
            code_a,
            code_b,
            "STATUS 200 (NFNS EDGE): Число 1 валидно. "
            "Delta = 0 признана штатной.",
        )

    # 3. Вычисление весов Хэмминга (PopCount) и дельты
    pop_a = sum(1 for bit in code_a.values() if bit == 1)
    pop_b = sum(1 for bit in code_b.values() if bit == 1)
    delta = pop_b - pop_a

    # 4. Штатный режим для всех остальных чисел
    if delta == 1:
        return (
            code_a,
            code_b,
            "STATUS 200: Данные валидны. Инвариант равен 1.",
```

```
    )

fixed_a = code_a.copy()
fixed_b = code_b.copy()

# 5. Локализация и восстановление по значению дельты
if delta == 2:
    # Угасание бита (1 -> 0) в Коде А; Код Б — эталон
    target_sum = processor_instance.decode(code_b)
    for pos in sorted(fixed_a.keys(), reverse=True):
        if fixed_a[pos] == 0:
            fixed_a[pos] = 1
            diff = abs(
                processor_instance.decode(fixed_a) - target_sum
            )
            if diff < 1e-9:
                return (
                    fixed_a,
                    fixed_b,
                    f"SEU FIX: Угасание бита в Коде А "
                    f"(позиция {pos}: 0 -> 1).",
                )
            fixed_a[pos] = 0

elif delta == 0:
    # Возгорание бита (0 -> 1) в Коде А; Код Б — эталон
    if is_zero_b:
        for pos, val in fixed_a.items():
            if val == 1:
                fixed_a[pos] = 0
                if all(v == 0 for v in fixed_a.values()):
                    return (
                        fixed_a,
                        fixed_b,
                        f"SEU FIX (ZERO): Возгорание бита в "
                        f"Коде А (позиция {pos}: 1 -> 0).",
                    )
                fixed_a[pos] = 1
    else:
        target_sum = processor_instance.decode(code_b)
        for pos in sorted(fixed_a.keys(), reverse=True):
            if fixed_a[pos] == 1:
                fixed_a[pos] = 0
                diff = abs(
                    processor_instance.decode(fixed_a)
                    - target_sum
                )
                if diff < 1e-9:
                    return (
                        fixed_a,
                        fixed_b,
                        f"SEU FIX: Возгорание бита в Коде А "
                        f"(позиция {pos}: 1 -> 0).",
                    )
                fixed_a[pos] = 1
```

```
elif delta == -1:
    # Возгорание бита (0 -> 1) в Коде Б; Код А — эталон
    if is_all_zero_a:
        if fixed_b.get("Z") == 1:
            for pos, val in fixed_b.items():
                if pos != "Z" and val == 1:
                    fixed_b[pos] = 0
                    only_z = all(
                        v == 0
                        for k, v in fixed_b.items()
                        if k != "Z"
                    )
                    if only_z:
                        return (
                            fixed_a,
                            fixed_b,
                            f"SEU FIX (ZERO): Возгорание бита "
                            f"в Коде Б (позиция {pos}: 1 -> 0).",
                        )
                    fixed_b[pos] = 1
    else:
        target_sum = processor_instance.decode(code_a)
        for pos in sorted(fixed_b.keys(), reverse=True):
            if fixed_b[pos] == 1:
                fixed_b[pos] = 0
                diff = abs(
                    processor_instance.decode(fixed_b)
                    - target_sum
                )
                if diff < 1e-9:
                    return (
                        fixed_a,
                        fixed_b,
                        f"SEU FIX: Возгорание бита в Коде Б "
                        f"(позиция {pos}: 1 -> 0).",
                    )
                fixed_b[pos] = 1

elif delta == 3:
    # Угасание бита (1 -> 0) в Коде Б; Код А — эталон
    target_sum = processor_instance.decode(code_a)
    for pos in sorted(fixed_b.keys(), reverse=True):
        if fixed_b[pos] == 0:
            fixed_b[pos] = 1
            diff = abs(
                processor_instance.decode(fixed_b) - target_sum
            )
            if diff < 1e-9:
                return (
                    fixed_a,
                    fixed_b,
                    f"SEU FIX: Угасание бита в Коде Б "
                    f"(позиция {pos}: 0 -> 1).",
                )
            fixed_b[pos] = 0
```

```
# Критический множественный сбой
raise RuntimeError(
    f"HARDWARE EXCEPTION: Критическое разрушение "
    f"дуального базиса! Delta = {delta}"
)
```

```
def nfns_edge_guard(
    code_a: dict, code_b: dict, reg_mode: int
) -> bool:
    """
    Краевой предохранитель для целочисленного режима NFNS
    (REG_MODE == 0). Возвращает True, если на входе штатное
    дуальное представление числа 1 (разряды F1 и F2), при
    котором разность весов (Delta = 0) является нормой.
    """
    if reg_mode != 0:
        return False
    if code_b.get("Z", 0) != 0:
        return False

    is_one_a = (
        code_a.get(1) == 1
        and code_a.get(2) == 0
        and all(v == 0 for k, v in code_a.items() if k not in (1, 2))
    )
    is_one_b = (
        code_b.get(1) == 0
        and code_b.get(2) == 1
        and all(
            v == 0 for k, v in code_b.items() if k not in (1, 2, "Z")
        )
    )
    return is_one_a and is_one_b
```