PAPER Special Section on Solid-State Circuit Design — Architecture, Circuit, Device and Design Methodology

Non-Stop Microprocessor for Fault-Tolerant Real-Time Systems

Shota NAKABEPPU[†a)], *Student Member and* **Nobuyuki YAMASAKI**[†b)], *Member*

SUMMARY It is very important to design an embedded real-time system as a fault-tolerant system to ensure dependability. In particular, when a power failure occurs, restart processing after power restoration is required in a real-time system using a conventional processor. Even if power is restored quickly, the restart process takes a long time and causes deadline misses. In order to design a fault-tolerant real-time system, it is necessary to have a processor that can resume operation in a short time immediately after power is restored, even if a power failure occurs at any time. Since current embedded real-time systems are required to execute many tasks, high schedulability for high throughput is also important. This paper proposes a non-stop microprocessor architecture to achieve a fault-tolerant real-time system. The non-stop microprocessor is designed so as to resume normal operation even if a power failure occurs at any time, to achieve little performance degradation for high schedulability even if checkpoint creations and restorations are performed many times, to control flexibly non-volatile devices through software configuration, and to ensure data consistency no matter when a checkpoint restoration is performed. The evaluation shows that the non-stop microprocessor can restore a checkpoint within $5\mu sec$ and almost hide the overhead of checkpoint creations. The non-stop microprocessor with such capabilities will be an essential component of a fault-tolerant real-time system with high schedulability.

key words: *non-volatile flip-flop, non-volatile memory, embedded real-time system, fault-tolerant system, microprocessor architecture*

1. Introduction

Many embedded real-time systems such as automobiles, robots, spacecraft, wearable devices, and sensor networks are social infrastructures in modern society. Such embedded real-time systems must be designed as fault-tolerant real-time systems with dependability. In particular, when a power failure occurs in an embedded real-time system using a conventional processor, as all flip-flops and the main memory values are lost, a restart process is required after power is restored. Even if power is restored quickly, the restart process takes a long time and causes deadline misses. A fault-tolerant real-time system requires a processor that can resume operation in a short time immediately after power is restored, no matter when a power failure occurs. Checkpointing with non-volatile devices is one effective method to achieve such a processor. Since current embedded real-time systems need to perform many tasks, high schedulability for high throughput is also important. A real-time system should be able to be scheduled by a real-time scheduler, including the overhead of checkpoint creations and restorations. The significant overhead of checkpoint creations and restorations reduces the number of tasks scheduled within a time constraint, reducing schedulability.

This paper proposes a non-stop microprocessor architecture for a fault-tolerant real-time system. The non-stop microprocessor can resume normal operation even if a power failure occurs at any time, can achieve little performance degradation for high schedulability even if checkpoint creations and restorations are performed many times, can flexibly control non-volatile devices through software configuration, and can ensure data consistency no matter when a checkpoint restoration is performed. Specifically, the non-stop microprocessor significantly reduces the overhead of the restart process by making the processor pipeline non-volatile and creating checkpoints on non-volatile devices. The non-stop microprocessor is designed so as to create two checkpoint systems to ensure a normal checkpoint restoration, even if a power failure occurs at any time. The non-stop microprocessor achieves little performance degradation for high schedulability even if many checkpoint creations and restorations are performed by significantly reducing the overhead of checkpoint creations and restorations and almost hiding the overhead of checkpoint creations. We design a non-volatile device controller architecture that can flexibly control non-volatile devices through software configuration in order to support a wide variety of non-volatile devices. We also design a consistency-aware data cache to ensure data consistency no matter when a checkpoint restoration is performed.

The contribution of this paper is to design a control architecture for various non-volatile devices to realize a non-stop microprocessor that can operate normally immediately after power is restored, even if the power is turned off at any point in time, with little performance degradation. Furthermore, we have designed the non-stop microprocessor based on the proposed control architecture.

2. Background

2.1 Checkpointing-Aware Real-Time Scheduling

Checkpointing-aware real-time scheduling has been studied to realize fault-tolerant real-time systems [1], [2]. Figure 1 shows an example of checkpointing-aware real-time scheduling. In this example, a real-time task with a time

Manuscript received August 3, 2022.
Manuscript revised November 6, 2022.
Manuscript publicized January 25, 2023.

[†]The authors are with the Department of Information and Computer Science, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: beppu@ny.ics.keio.ac.jp
b) E-mail: yamasaki@ny.ics.keio.ac.jp

DOI: 10.1587/transele.2022CDP0005

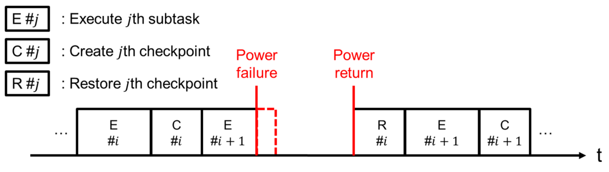

E #j : Execute jth subtask
C #j : Create jth checkpoint
R #j : Restore jth checkpoint
Power failure
Power return
E #i
C #i
E #i + 1
R #i
E #i + 1
C #i + 1
t

Fig. 1 An example of checkpointing-aware real-time scheduling

constraint is divided into multiple subtasks, and checkpoints are created periodically. When a power failure occurs, the operation resumes immediately after a power return by restoring the most recent and appropriate checkpoint. In order to cope with a power failure, it is necessary to create and restore sufficient checkpoints using non-volatile devices.

In checkpointing-aware real-time scheduling, the WCET (Worst-Case Execution Time) is calculated based on the assumption that transient faults (e.g., a power failure and single event upsets [3]) occur at most k times, taking into account the overhead of checkpoint creation and restoration, so that time constraints can be guaranteed even if transient faults occur m ($m <= k$) times [1], [2], [4]. Since the scheduling can tolerate burst faults, the scheduling is useful even if a power failure equal to n ($n <= k$) times faults occurs. Since current embedded real-time systems require high schedulability, the WCET, including the overhead of checkpoint creation and restoration, should be as short as possible by reducing the overhead [4].

A fault-tolerant real-time system requires that the system can resume normal operation as soon as power is restored, no matter when a power failure occurs. If a fault-tolerant system creates a single checkpoint, the normal checkpoint is lost if a power failure occurs during checkpoint creation. If multiple checkpoints are created with slightly different creation times, the normal checkpoints can be retained even if a power failure occurs during checkpoint creation. Therefore, to realize a fault-tolerant real-time system with high schedulability, it is necessary to create and restore checkpoints using non-volatile devices, reduce or hide the overhead of checkpoint creations and restorations, and create multiple systems of checkpoints.

2.2 Non-Volatile Memory (NVM)

Non-Volatile Memory (NVM) is a memory that can retain saved values even when power is turned off. Typical processors for embedded systems equip volatile memory, such as a DRAM and an SRAM, as the main memory and non-volatile memory, such as flash memory, as the secondary memory. In recent years, processors with a non-volatile memory such as an MRAM [5], a FeRAM [6], a ReRAM [7], and a PCM [8], which enable high-speed reading and writing as the main memory have also been studied. When a processor equips a non-volatile memory as the main memory, the values stored in the main memory are retained even when the power is turned off so that values stored in the main memory can be excluded from a checkpoint.

2.3 Non-Volatile Flip-Flop (NVFF)

A typical Flip-Flop (FF) is a volatile device whose value is lost after the power is turned off. A Non-Volatile Flip-Flop (NVFF) [9]–[15] consists of an FF and a non-volatile element. The data stored in the FF is volatile, and the data stored in the non-volatile element is non-volatile. The NVFF can store the value of the FF in the non-volatile element or restore the value of the non-volatile element to the FF. The operation of storing the FF's value in the non-volatile element is called *a store operation*. The operation of restoring the non-volatile element's value to the FF is called *a restore operation*. The NVFF has control signal pins to perform a store operation and a restore operation at the intended timing. Note that the FF's value and the non-volatile element's value are updated at independent times. On the one hand, the NVFF stores input data into the FF at the rising edge of a clock, as with a typical FF. On the other hand, the NVFF stores the FF's value into the non-volatile element only when a store operation is performed. Since a store operation fails with a certain probability [15], an SoC with many NVFFs must have a function to verify whether each NVFF has successfully performed a store operation. Therefore, a VR-NVFF [15] is useful for such SoCs because it can perform *a verify operation* that compares whether the non-volatile element's value is equal to the FF's value. A VR-NVFF equips two Magnetic Tunneling Junctions (MTJs) [16]. Note that a store operation stores the FF's 1-bit value into the MTJ pair.

2.4 Non-Volatile Processor (NVP)

Non-Volatile Processors (NVP) that can create checkpoints on non-volatile devices have been studied [17]–[19]. The NVPs can resume normal operation immediately after power restoration by restoring the most recent and appropriate checkpoints in the event of a power failure. Most existing researches focus on reducing power consumption, and there is no NVP suitable for a fault-tolerant real-time system with high schedulability.

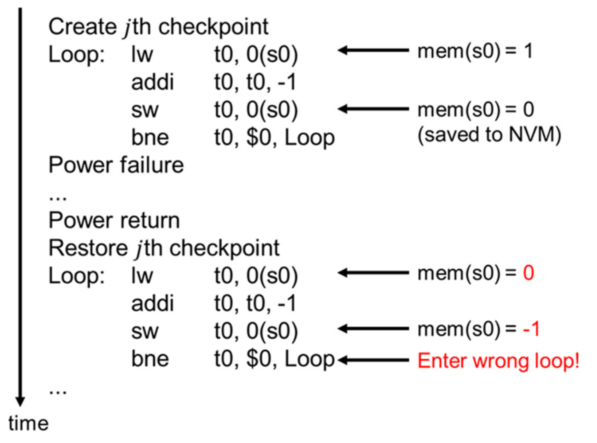
On the one hand, the NVPs that can create checkpoints with NVM have been studied. In these NVPs, two main methods of checkpoint creation have been proposed. One method is to use a volatile memory as the main memory and copy the values of the main memory and FFs to an NVM [20]. The other method is to use an NVM as the main memory and copy only the values of FFs to an NVM [21]–[23]. These NVPs can create multiple checkpoints as long as the NVM capacity allows. The most recent and appropriate checkpoint can be restored even if a power failure occurs during a checkpoint creation. However, a checkpoint creation and a checkpoint restoration require many NVM read/write operations, and the overhead of a checkpoint creation and a checkpoint restoration is significant. Therefore, although these methods can resume normal operation after the power is restored, even if a power failure occurs at any time, the significant overhead of checkpoint creations and restorations degrades schedulability in real-time systems. An NVP scheduler [24] has been studied to improve the schedulability of systems using NVP, but the overhead of checkpoint creations and restorations still needs to be addressed.

On the other hand, the NVPs that replace FFs with NVFFs and create checkpoints with NVFFs also have been studied [25]–[27]. These NVPs can reduce the overhead of checkpoint creations and restorations by performing store and restore operations in parallel on each NVFF. However, this method can create only one checkpoint because each NVFF can store only a one-bit value in its non-volatile device. If a power failure occurs during a checkpoint creation, a checkpoint is no longer normal because only some of a checkpoint is updated, and the rest is not updated. Therefore, although this method can reduce the overhead of checkpoint creations and restorations, it cannot resume normal operation after power is restored if a power failure occurs during a checkpoint creation.

Thus, highly schedulable fault-tolerant real-time systems cannot be realized using these NVPs. We design a non-stop microprocessor that can resume normal operation immediately after power is restored, even if a power failure occurs at any time while reducing the overhead of checkpoint creations and restorations. In order to achieve these requirements, the microprocessor is designed to use an NVM as main memory, replace FFs with NVFFs, and create two sets of checkpoints.

2.5 Consistency-Aware Checkpoint Creation

In architectures that equip an NVM and NVFFs and use an NVM as main memory, a power failure may cause data inconsistency because an NVM and NVFFs have different timings for writing values to non-volatile devices. In the case of an NVM, all written values are non-volatile, while in the case of an NVFF, the FF's value is volatile, and only the non-volatile device's value is non-volatile. Figure 2 shows an example of data inconsistency after a power



```
Create jth checkpoint
Loop:   lw      t0, 0(s0)      ←—— mem(s0) = 1
        addi    t0, t0, -1
        sw      t0, 0(s0)      ←—— mem(s0) = 0
        bne     t0, $0, Loop        (saved to NVM)
Power failure
...
Power return
Restore jth checkpoint
Loop:   lw      t0, 0(s0)      ←—— mem(s0) = 0
        addi    t0, t0, -1
        sw      t0, 0(s0)      ←—— mem(s0) = -1
        bne     t0, $0, Loop   ←—— Enter wrong loop!
...
```

Fig. 2 An example of data inconsistency caused by power failure

failure occurs. In order to guarantee data consistency even if a power failure occurs at any time, the program analysis to insert checkpoint creations that guarantee data consistency has been studied [28]–[32]. A checkpoint creation to guarantee data consistency is called a *consistency-aware checkpoint creation*. Inserting consistency-aware checkpoint creations can ensure data consistency but increases overhead by creating many checkpoints. Therefore, some hardware-based approaches to reduce the overhead of consistency-aware checkpoint creations have been proposed. PROWL [33] is a consistency-aware cache replacement policy to avoid data inconsistency with fewer checkpoint creations. COACH [34] and the proposed NVP by Senni et al. [35] equips two systems of NVMs to resolve data inconsistencies fundamentally without a consistency-aware checkpoint creation.

We designed a consistency-aware data cache to guarantee data consistency with as few consistency-aware checkpoint creations as possible, no matter when a power failure occurs.

3. Non-Stop Microprocessor

3.1 Overall Architecture

This section describes the architecture of the non-stop SoC (NVIOC SoC: Non-Volatile IO Core System-on-Chip). The NVIOC SoC integrates a non-stop microprocessor designed using non-volatile devices (NVIOC), a clock control module, memories (embedded MRAM and SRAM), and required IO peripherals (GPIO, UART, SPI, I2C, PWM generators, and pulse counters) with DMACs. The baseline CPU architecture of NVIOC is a simple RISC processor (IOC: IO Core processor). The NVIOC is designed by replacing all the flip-flops used in the IOC's pipeline with NVFFs to make it non-volatile at the pipeline level. In order to eliminate checkpointing for values in the main memory, an NVM, such as an MRAM, is used as the main memory so that the main memory values are non-volatile. Since embedded real-time systems often perform actuator control, restoring the state of the CPU and the IOs used for motor control is desirable. Therefore, the pulse counter and the PWM input modules are also designed as non-stop IOs using NVFFs. The PWM output module is one of the IOs for motor control. However, it is not designed as a non-stop IO because its state can be immediately restored by recalculation based on the values of the PWM input module and the pulse counter module. The block diagram of the NVIOC SoC is shown in Fig. 3. The blue parts are volatile, and the yellow parts are non-volatile.

The NVFFM (Non-Volatile Flip-Flop Module) is a register file that consists of NVFFs and a control logic that performs a checkpoint creation and a checkpoint restoration. The NVDC (Non-Volatile Device Controller) is designed to control NVFFMs of various configurations with different bit widths and flexibly absorb and control the differences in configuration through software settings. The NVIOC

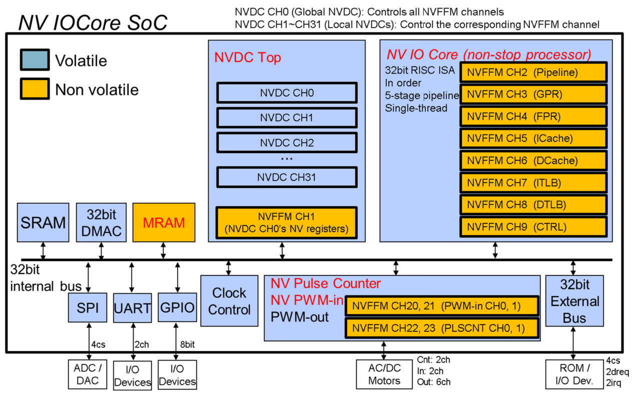


Fig. 3 The figure shows the block diagram of the NVIOC SoC.

SoC has multiple NVFFM and NVDC channels, and each NVFFM channel is controlled by its corresponding NVDC channel.

3.2 NVFFM (Non-Volatile Flip-Flop Module)

A fault-tolerant real-time system should operate normally, no matter when a power failure occurs. Therefore, we design the NVFFM to maintain normal checkpoints at any time. The NVFFM creates a checkpoint on non-volatile devices so that the checkpoint will not be lost even if a power failure occurs. On the one hand, if the NVFFM creates a single checkpoint, the normal checkpoint is lost if a power failure occurs during a checkpoint creation. On the other hand, if the NVFFM creates multiple checkpoints with slightly different creation times, the normal checkpoints can be retained at any time. Therefore, the NVFFM is designed to have two NVFFs for each 1-bit value (NVFF pair) to create two checkpoints with slightly different creation times. In this paper, one of the NVFF pairs is called a bank (e.g., bank0 and bank1). The NVFFM has a valid bit in order to indicate which bank contains a normal and latest checkpoint. For example, if a valid bit is 1, bank1 has the normal and latest checkpoint. Since the value of a valid bit is required for a checkpoint restoration after a power failure occurs, the valid bit is stored in an NVFF. A fault-tolerant real-time system should achieve little performance degradation for high schedulability even with frequent checkpoint creations and restorations. Therefore, we design the NVFFM to significantly reduce the overhead of checkpoint creations and restorations by allowing multiple NVFFs to perform store or restore operations in parallel. In addition, the NVFFM hides the overhead of checkpoint creations by alternating store operations with two banks. A fault-tolerant real-time system should operate normally despite various failures, such as permanent MTJ failures and failures of store operations. Therefore, we design the NVFFM with VR-NVFFs [15] so that we can verify the success or failure of a store operation to ensure that the non-stop microprocessor with many NVFFs operates properly. In addition, the NVFFM supports error correction code (ECC) encoding and

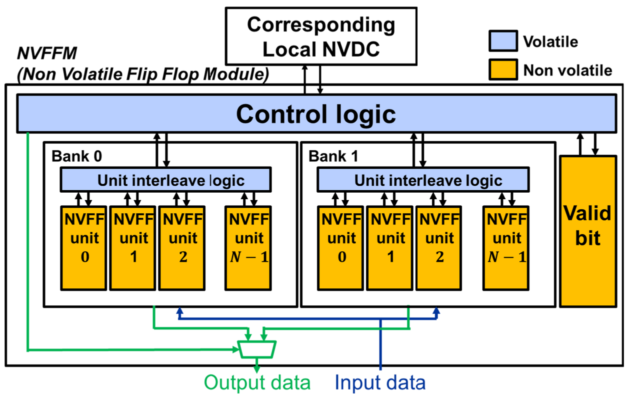


Fig. 4 The NVFFM has 2-bank. Input data is assigned to both banks. Output data is output from the bank selected by the control logic. Even while one bank is performing a checkpoint creation, the other bank can be used as a simple register file. The valid bit indicates which bank has a valid checkpoint.

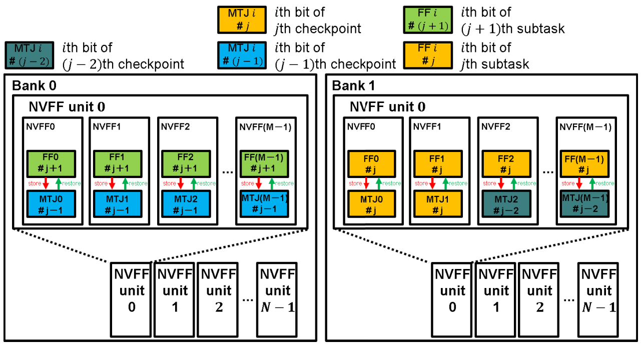

MTJ i #(j − 2) ith bit of (j − 2)th checkpoint
MTJ i #j ith bit of jth checkpoint
MTJ i #(j − 1) ith bit of (j − 1)th checkpoint
FF i #(j + 1) ith bit of (j + 1)th subtask
FF i #j ith bit of jth subtask
Bank 0
Bank 1
NVFF unit 0
NVFF0
NVFF1
NVFF2
NVFF(M−1)
NVFF unit 0
NVFF unit 1
NVFF unit 2
NVFF unit N − 1

Fig. 5 The figure shows the NVFF banks when the NVFFM is creating the jth checkpoint.

decoding to cope with permanent MTJ failures. A (64,72) Hamming code is used as ECC to perform 1-bit error correction and 2-bit error detection. Figure 4 shows the block diagram of the NVFFM. Figure 5 shows the NVFF banks when the NVFFM is creating the jth checkpoint. While the bank1 is creating jth checkpoint, the bank0 is performing as a simple register file for $j+1$th subtask. On the one hand, the NVFF0 and NVFF1 in the NVFF unit0 of the bank1 have already stored the jth checkpoint information into the MTJs. On the other hand, the other NVFFs in the bank1 still store the $j-2$th checkpoint information into the MTJs. Therefore, if a power failure occurs at this time, the checkpoint information stored in the bank1 is inconsistent. Since the bank0 has already stored the $j-1$th checkpoint into the MTJs, the NVFFM can restart $j-1$th checkpoint information even in this case.

3.2.1 Normal Operation

The NVFFM operates as a simple N-bit register file in case of a normal operation. Each NVFF stores input data into the FF at the rising edge of a clock, as with a typical FF. In order to reduce energy consumption, the NVFFM selects only either bank to perform read and write operations. The selected bank is called the *active bank*.

3.2.2 Checkpoint Creation with an NVFFM

We explain how to perform a checkpoint creation with an NVFFM.

1. The NVFFM selects which bank to create the checkpoint. The selected bank is called the *store target bank*. Suppose a checkpoint creation is performed on the bank indicated by the value of a valid bit. In that case, the normal and latest checkpoint will be lost if a power failure occurs. Therefore, the NVFFM selects the bank indicated by the inverted value of a valid bit. For example, if a valid bit is 1, the *store target bank* is bank0.
2. The NVFFM sends input data to both banks. Each NVFF of both banks stores input data into the FF, as with a typical FF. Since both banks have the latest input data, the NVFFM can operate normally regardless of which bank is selected as the *active bank*.
3. The NVFFM selects the bank indicated by the valid bit value as the *active bank*. Although the *store target bank* cannot be accessed during the checkpoint creation, the other bank can be accessed during the checkpoint creation. Therefore, the NVFFM can continue to operate as a simple N-bit register file even while the checkpoint creation by selecting the other bank as the *active bank*. This is why the NVFFM can hide the overhead of the checkpoint creation.
4. The NVFFM encodes ECC based on the FFs' values of the NVFFs in the *store target bank*.
5. The clock control unit gates the clock of the *store target bank*. Therefore, the FF's value of each NVFF in the *store target bank* does not change during the checkpoint creation.
6. The NVFFM performs a verify operation on each NVFF in the *store target bank*. If the FF's value and the MTJs' value are the same for all NVFFs, the NVFFM skips the seventh and eighth stages because there is no need to perform a store operation.
7. Each NVFF unit in the *store target bank* operates NVFFs' control signals to perform a store operation.
8. Each NVFF unit in the *store target bank* operates NVFFs' control signals to perform a verify operation. If the MTJ's value and the FF's value are the same for all NVFFs, the NVFFM goes to the ninth stage. If the MTJ's value and the FF's value are different for an NVFFs, the NVFFM checks *store retry counter*. If the value of *store retry counter* is not larger than the user-defined maximum value, the NVFFM increments *store retry counter* and goes to the seventh stage. If the value of *store retry counter* is larger than the user-defined maximum value, the NVFFM aborts the checkpoint creation and generates interrupts to handle store failures.
9. The NVFFM updates the value of a valid bit to indicate the *store target bank*. The updated value is written to the FF of the NVFF, which contains a valid bit. If a power failure occurs until this stage, the value of a valid bit is not stored in the MTJ of the NVFF. Therefore, the NVFFM can perform a checkpoint restoration with the bank, which has a normal checkpoint.
10. The NVFFM performs a store operation on the NVFF, which contains a valid bit. If a power failure occurs at this stage, the MTJ's value may become unstable. Therefore, we cannot know whether a valid bit will become '0' or '1' on a checkpoint restoration. This seems like a critical problem, but it is not. Since both banks have a normal checkpoint at this stage, the NVFFM can perform a checkpoint restoration normally regardless of which bank is used. This ten-step process allows the NVFFM to maintain a normal checkpoint no matter when a power failure occurs while hiding the overhead of a checkpoint creation.

The NVFFM supports *Store Only (SO)*, *One Step Store (OSS)* [15], *Two Step Store (TSS)* [15], and *Multi Step Store (MSS)* methodologies for a checkpoint creation. Figure 6 shows the state flows of each methodology. We call a checkpoint creation with *SO* method as *SO checkpointing*. SO checkpointing must perform store operations on many NVFFs. It also has to perform a *long store operation* which takes a long enough time for all NVFFs to successfully store the FF's value to the MTJ in a single store operation. Therefore, SO checkpointing consumes much power. We call a checkpoint creation with *OSS* method as *OSS checkpointing*. OSS checkpointing reduces power consumption by reducing the number of NVFFs that perform store operations. The NVFFM performs verify operations before performing store operations so that only NVFFs for which the MTJ's value differs from the FF's value are subject to the store operations. Although, OSS checkpointing must perform *long store operation*. We call a checkpoint creation with *TSS* method as *TSS checkpointing*. TSS checkpointing reduces power consumption by reducing the time spent on the first store operation. It takes advantage of the fact that most NVFFs can write the FF value to the MTJ successfully, even with a store operation in a short time. The first store operation is *short store operation* which takes a short time, and the second store operation is *long store operation*. We propose *MSS* method as an extension of *TSS* method for more fault tolerance. We call a checkpoint creation with *MSS* method as *MSS checkpointing*. The NVFFM repeats the verify and store operations until all NVFFs successfully write the FF's values to the MTJs. If the number of verify operations exceeds the user-defined maximum value, the NVFFM generates an interrupt and terminates a checkpoint creation. MSS checkpointing allows users to respond to failures of a checkpoint creation by handling the interrupt.

Figure 7 shows an example of checkpointing-aware real-time scheduling with an NVFFM. The overhead of a checkpoint creation is almost hidden because subtasks are

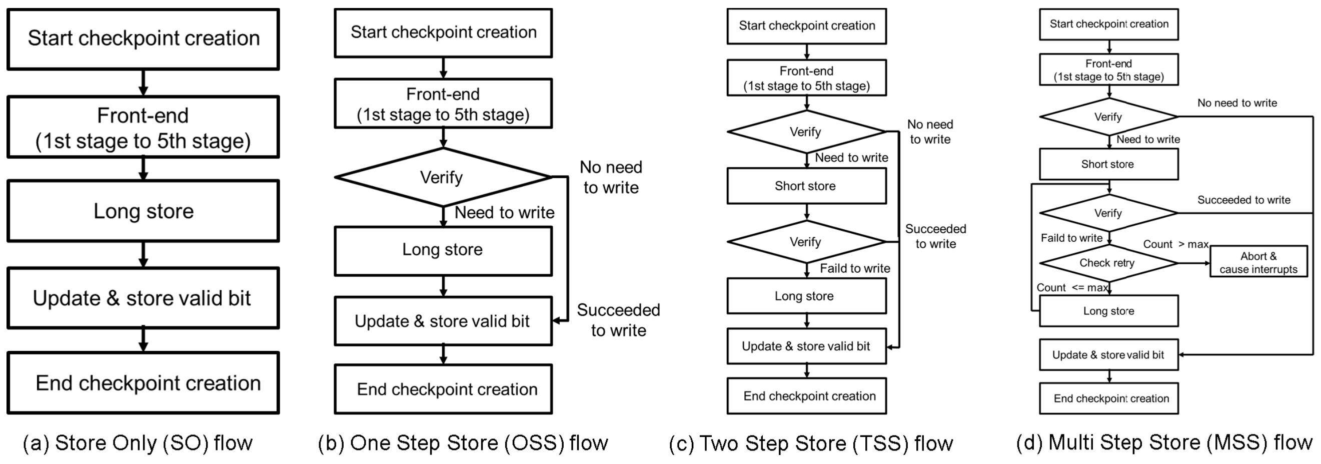


Fig. 6 The figure shows the state flows of *Store Only (SO)*, *One Step Store (OSS)* [15], *Two Step Store (TSS)* [15], and *Multi Step Store (MSS)* methodologies for a checkpoint creation.

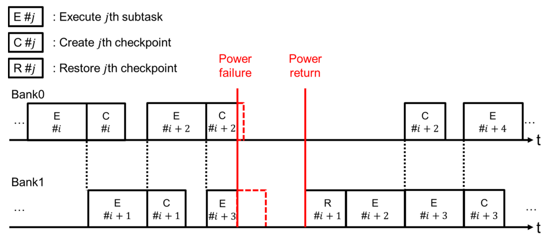

E #j : Execute jth subtask
C #j : Create jth checkpoint
R #j : Restore jth checkpoint
Power failure
Power return
Bank0
Bank1
t

Fig. 7 The figure shows an example of checkpointing-aware real-time scheduling with an NVFFM.

executed with one bank, even while a checkpoint creation is performed with the other bank. Since at least one bank always has a normal checkpoint, the microprocessor can resume the normal checkpoint immediately after power is restored, even if a power failure occurs during a checkpoint creation.

3.2.3 Checkpoint Restoration with an NVFFM

We explain how to perform a checkpoint restoration with an NVFFM.

1. The NVFFM performs a restore operation on the NVFF that contains a valid bit to restore the MTJ's value to the FF.
2. The NVFFM checks a valid bit to select which bank has a normal and latest checkpoint. The selected bank is called the *restore target bank*. For example, if a valid bit is 1, the *restore target bank* is bank1.
3. Each NVFF unit in the *restore target bank* operates NVFFs' control signals to perform a restore operation.
4. The NVFFM decodes ECC based on the FFs' values of the NVFFs in the *restore target bank*. If 1-bit error correction or 2-bit error detection occurs, the NVFFM generates the corresponding interruption to handle them.
5. The NVFFM selects the *restore target bank* as the *active bank*.
6. The NVFFM acts as an N-bit simple register file using the *active bank*. This six-stage process allows the NVFFM to perform a checkpoint restoration with the normal and latest checkpoint.

3.2.4 Checkpoint Creation and Restoration with Multiple NVFFMs

We have discussed the case of a checkpoint creation and a checkpoint restoration with an NVFFM. However, the multiple NVFFMs cause problems when they perform a checkpoint creation and a checkpoint restoration. Since the time to complete a checkpoint creation on each NVFFM is different, the time to complete the store operation on each NVFFM's valid bit is different. Suppose a power failure occurs during a checkpoint creation. In that case, some NVFFMs may perform a checkpoint restoration with bank0 and others with bank1, and the entire system may not restore checkpoints normally. Therefore, we designed a *global bank selection mode* such that the NVFFM select one of the NVFFMs as the *master NVFFM* and each NVFFM detects the bank that has the normal and latest checkpoint using the valid bit of the *master NVFFM*. The valid bit of the *master NVFFM* is called the *global valid bit*. Since the value of a *global valid bit* is required for a checkpoint restoration after a power failure occurs, the *global valid bit* is stored in an NVFF. In the *global bank selection mode*, the value of the *global valid bit* is stored in the MTJ after all NVFFMs complete checkpoint creations. Therefore, the entire system can restore checkpoints normally even if a power failure occurs during a checkpoint creation. Specifically, the master NVFFM performs a checkpoint creation after all NVFFMs except the master NVFFM complete checkpoint creations. This mode guarantees that all NVFFMs perform checkpoint restorations with the same bank.

An MTJ-based NVFF, such as a VR-NVFF, consists of an MTJ circuit and a flip-flop circuit. On the one hand, A flip-flop circuit is stable in operation. On the other hand, an

MTJ circuits has a certain probability of failure. Therefore, the MTJ circuit of the NVFF that stores the *global valid bit* becomes a single point of failure. In order to address this problem, we designed a *software bank selection mode* such that each NVFFM selects the bank using an address-mapped control register. The control register that selects the bank is called *bank selection register*. In the *software bank selection mode*, software flexibly controls each NVFFM using the management information of each NVFFM located on non-volatile memory. The *bank select register* is updated based on the management information when each NVFFM performs a checkpoint creation and a checkpoint restoration. The management information can be located at any address on the non-volatile memory, thus eliminating single points of failure. On the other hand, software overhead increases the overhead of checkpoint creations and restorations.

Although we are also considering another method that introduces a multi-bit majority voting circuit and an error correction mechanism to remove single points of failure, they will be introduced in future work.

3.3 NVDC (Non-Volatile Device Controller)

We design a global NVDC (NVDC channel 0) to control multiple NVFFMs in a short time and local NVDCs (other NVDC channels) to control each NVFFM individually. Global NVDC can control multiple NVFFMs simultaneously. On the other hand, local NVDCs can control only its corresponding NVFFMs (e.g., NVDC channel P ($1 <= P <= 31$) controls only NVFFM channel P).

Figure 8 shows the block diagram of the global NVDC. The values of non-volatile control registers are saved to NVFFM CH1 so as to retain essential configurations even if a power failure occurs. The global NVDC has its exclusive interrupt controller that can handle interrupts from the NVM. The global NVDC can initiate a checkpoint creation or restoration on multiple NVFFMs via address-mapped control registers or external control signals from IOs. The global NVDC can control the control logic of arbitrary local NVDCs simultaneously so as to control multiple NVFFMs in parallel.

Figure 9 shows the block diagram of the local NVDCs. Each local NVDC has its exclusive interrupt controller that can handle interrupts from its corresponding NVFFM. Each local NVDC can initiate a checkpoint creation or restoration on its corresponding NVFFM via address-mapped control registers. Each local NVDC can control whether to perform ECC encoding and decoding on its corresponding NVFFM, considering the trade-off between fault-tolerance and power consumption. Each local NVDC can select the checkpoint creation method on its corresponding NVFFM from *Store Only (SO)*, *One Step Store (OSS)* [15], *Two Step Store (TSS)* [15], and *Multi Step Store (MSS)* methodologies. As described below, each local NVDC has functions such as Controlling the switching time of NVFF's control signals, unit-interleave function, and auto-store function. These functions allow the NVDCs to provide flexible control for various situations through software settings.

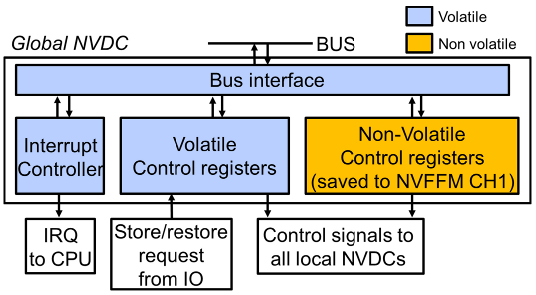

Volatile
Non volatile
Global NVDC
BUS
Bus interface
Interrupt Controller
Volatile Control registers
Non-Volatile Control registers (saved to NVFFM CH1)
IRQ to CPU
Store/restore request from IO
Control signals to all local NVDCs

Fig. 8 The global NVDC has the interrupt controller and can output interrupt requests (IRQs) to the CPU. Store/restore requests from IO can initiate a checkpoint creation/restoration. The global NVDC can control multiple local NVDCs simultaneously.

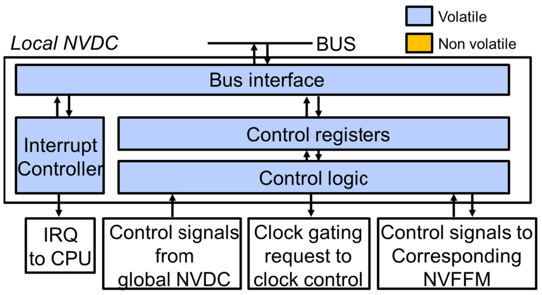

Volatile
Non volatile
Local NVDC
BUS
Bus interface
Interrupt Controller
Control registers
Control logic
IRQ to CPU
Control signals from global NVDC
Clock gating request to clock control
Control signals to Corresponding NVFFM

Fig. 9 The local NVDC has an interrupt controller, which can output interrupt requests (IRQs) to the CPU. The local NVDC controls its corresponding NVFFM.

3.3.1 Control Switching Time of NVFF's Control Signals

A typical NVFF correlates between the time of switching control signals and the probability of successfully storing a value in the MTJ [15]. It will be impossible to create a checkpoint if the switching time is set to a fixed number of clock cycles of the NVFFM's operating frequency and a store operation does not work well within the switching time. In addition, if a control method that dynamically changes the frequency, such as DVFS, is used, the switching time may vary due to changes in the clock frequency, making it impossible to successfully store the value in the MTJ. In order to cope with this situation, the NVDCs can control the number of clock cycles required to switch NVFF's control signals, considering the trade-off between the checkpoint creation or restoration overhead and the probability of successfully storing or restoring.

3.3.2 Unit-Interleave Function

The fastest way of a checkpoint creation or a checkpoint restoration is to perform a store or restore operation on NVFFs simultaneously. However, this method may not work correctly due to the high peak current. To reduce peak current, the NVFFM's bank is divided into *NVFF units* that consist of multiple NVFFs as Fig. 4. Furthermore, *a* *unit-interleaving function* is designed to control the number of clock cycles (C_S or C_R) from the start of one unit's store or restore operation to the start of the next unit's store or restore operation. As C_S or C_R is increased, peak power can be reduced by reducing the maximum number of NVFFs performing simultaneous store or restore operations while the overhead of a checkpoint creation or a checkpoint restoration increases. The NVDCs can control each NVFFM's C_S or C_R by the unit-interleave function, considering the trade-off between the checkpoint creation or restoration overhead and the peak current.

3.3.3 Auto-Store Function

Although the NVDCs can initiate a checkpoint creation and a checkpoint restoration via address-mapped control registers, this method includes software overhead. In particular, a checkpoint creation using this method degrades performance because checkpoint creations are performed frequently. In order to avoid performance degradation by checkpoint creations, the NVDCs support *auto-store function*, which is fully hardware-based automatic and periodic checkpoint creations. *Auto store function* has a dedicated hardware timer and creates a checkpoint each time the timer value reaches a user-defined value. Since this function is performed entirely in hardware, the function does not degrade performance.

3.4 Consistency-Aware Data Cache

In order to guarantee consistency, a consistency-aware checkpoint creation is required before a value is written to the NVM. The data cache cannot write data to the NVM until the consistency-aware checkpoint creation is complete. On the one hand, as long as the write buffer is free, the data cache can continue to operate. On the other hand, when the write buffer is full, the data cache stalls, and the microprocessor processing stops.

We propose two methods to address the consistency problem. The first method is to have the hardware generate an interrupt when it detects that a consistency-aware checkpoint is needed. Then, the interrupt calls a checkpointing function to create a checkpoint. This method allows a programmer to create a checkpoint at the intended time. However, the overhead of a consistency-aware checkpoint creation becomes significant because the interruption handling and the checkpointing function take a long time.

The data cache stalls when the write buffer is full, and the microprocessor stops processing. This makes it difficult to estimate the WCET and challenging to schedule with real-time scheduling. The second method is to create a checkpoint automatically by hardware. When the hardware detects that a consistency-aware checkpoint creation is required, it automatically creates a checkpoint. In this method, WCET changes little because the hardware immediately and automatically performs a consistency-aware checkpoint creation. However, a programmer cannot know when a consistency-aware checkpoint creation occurs. An unintended consistency-aware checkpoint creation could result in unintended behavior when a checkpoint is restored. Because of the negative effects of either method, hardware should be designed to reduce the number of consistency-aware checkpoint creations as much as possible.

We design the consistency-aware data cache to reduce the number of consistency-aware checkpoints as much as possible while ensuring consistency. The fewer writes to NVM, the fewer consistency-aware checkpoint creation are inserted. In order to reduce the number of writes to NVM as much as possible, a write-back/write-buffer/write-allocate format is employed. Furthermore, when a cache miss occurs in a store instruction, a typical write allocate cache writes data to the NVM and cache entry, but this design does not write data to the NVM but only to the cache entry. As a result, writing to NVM is only done when a write-back is required.

When a cache line requests a write-back, the data cache must determine if data consistency can be maintained by writing that entry to NVM. If the cache line is already stored in NVFFs of the data cache, data consistency is maintained even if the cache line is written to NVM. Otherwise, writing the entry to NVM may result in data inconsistency. We introduce *A stored bit* for data consistency check. Each cache line has its stored bit, and the stored bit determines if the cache line is acceptable for write-back.

When checkpoint creation is completed, the stored bit of each cache line whose dirty bit is one is set to one. Since cache lines whose stored bit is one are already stored in the MTJ of each NVFF, the memory system maintains consistency even when writing them to the NVM. If the stored bit is 0, the cache line is not stored in the MTJ of each NVFF. Therefore a consistency-aware checkpoint creation is required before writing them to NVM.

3.5 Requirements for the Non-Stop Processor Design

Our proposed non-stop processor design requires an NVM and an NVFF. This paper adopted an MRAM and a VR-NVFF for our non-stop processor design. However, any NVM and any NVFF can be adopted for the non-stop processor design. For example, a FeRAM and a ferroelectric-based NVFF can be adopted for the non-stop processor design. Note that the NVFFM functions, such as OSS checkpointing, TSS checkpointing, and MSS checkpointing, require a verify operation. If the adopted NVFF does not support a verify operation, the NVFFM cannot support OSS checkpointing, TSS checkpointing, and MSS checkpointing. Therefore, the NVFFM can create a checkpoint only with SO checkpointing. Unfortunately, SO checkpointing cannot detect a failure of a store operation. Since a store operation fails with a certain probability [15], we strongly recommend adopting an NVFF that supports a verify operation for the non-stop processor design. If the adopted NVFF supports a verify operation, the NVFFM can support all functions described in this paper. Since MSS checkpointing can detect a failure of a store operation, we can make the non-stop processor fault-tolerant.

4. Implementation

We have designed the NVIOC SoC using SONY's NVFFs and an MRAM with SONY 40nm process and taped out the NVIOC SoC. Figure 10 shows the chip layout of the NVIOC SoC.

The NVFF [15] is provided as a standard cell by SONY. The NVFFs are arranged as shown in the blue area of Fig. 10. Since the NVIOC SoC uses a large number of NVFFs (approximately 160k), some NVFFs may fail. Therefore, the NVFFM and NVDC were designed so as to provide error correction and error detection. The MRAM is located in the lower right corner and is densely wired for a power supply to provide sufficient power. We have been designing the NVIOC SiP (System-in-Package) that integrates the NVIOC SoC, DRAM, flash memory, FPGA, and power ICs, so as to implement the NVIOC SiP as soon as the NVIOC SoC is shipped.

5. Evaluation

5.1 Configuration

We evaluate the NVIOC SoC with RTL simulation using the Synopsys VCS simulator. We emulate the state at power off such that all reset signals are set to enable, and indefinite values are assigned to all NVFFs. The NVIOC SoC is equipped with SRAM and MRAM, and MRAM is used as the main memory in the evaluation. The evaluation parameters are shown in Table 1, Table 2, and Table 3. We define a *Pass_Rate* (*PR*) is the probability that an NVFF will successfully write the FF's value to the non-volatile element. The previous paper on an actual chip designed using VR-NVFFs [36] has evaluated the relationship between the time of a store operation and the pass rate. We determined the time of a long store operation (T_L), the time of a short store operation (T_S), and the time of a very short store operation (T_{VS}) considering the corresponding pass rate. T_L is set to a long enough time (140ns) for the pass rate on a long store operation (PR_L) to be around 1.00. T_S is set to a short time (40ns) under the condition that the pass rate on a short store operation (PR_S) is around 0.95 on average. T_{VS} is set to a very short time (20ns) under the condition that the pass rate on a very short store operation (PR_{VS}) is around 0.65 on average. We call TSS checkpointing whose first store operation time is T_{VS} as *TSS-VS checkpointing*, TSS checkpointing whose first store operation time is T_S as *TSS-S checkpointing*, MSS checkpointing whose first store operation time is T_{VS} as *MSS-VS checkpointing*, and MSS checkpointing whose first store operation time is T_S as *MSS-S checkpointing*. Restore energy, store current, and voltage in Table 3 refer to those of SSR-NVFF [14] because VR-NVFF is designed based on the SSR-NVFF. Since a verify

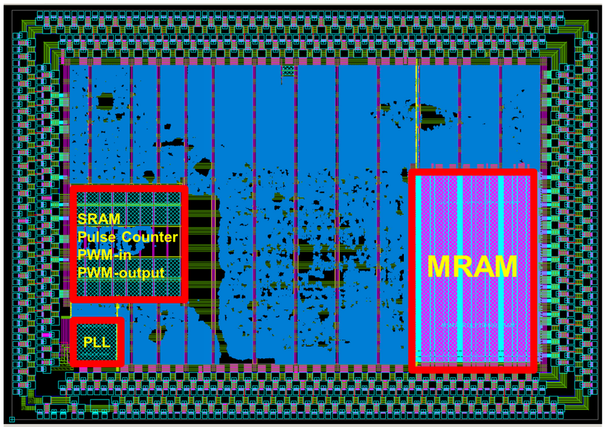


Fig. 10 This figure shows the chip layout of the NVIOC SoC. NVFFs are placed on the blue parts.

Table 1 NVIOC parameters

Instruction Cache	2KiB, 1-way, 64-set, VIPT
Data Cache	2KiB, 1-way, 64-set, VIPT, 16-entry write buffer, write back, write allocate
SRAM	256KiB
MRAM	3MiB
Frequency	100MHz

Table 2 NVFFM parameters

NVFFM Channel	Corresponding module	Units	NVFFs per unit
NVFFM CH1	Global NVDC	13	288
NVFFM CH2	Pipeline	6	288
NVFFM CH3	GPR	10	576
NVFFM CH4	FPR	10	576
NVFFM CH5	Instruction cache	22	1,152
NVFFM CH6	Data cache	27	1,152
NVFFM CH7	Instruction TLB	6	288
NVFFM CH8	Data TLB	6	288
NVFFM CH9	Control registers	17	288
NVFFM CH20	PWM in CH0	2	288
NVFFM CH21	PWM in CH1	2	288
NVFFM CH24	Pulse counter CH0	2	288
NVFFM CH25	Pulse counter CH1	2	288

Table 3 VR-NVFF parameters

The time of a long store operation (T_L)	140ns
The time of a short store operation (T_S)	40ns
The time of a very short store operation (T_{VS})	20ns
The pass rate on a long store operation (PR_L)	1.00
The pass rate on a short store operation (PR_S)	0.95
The pass rate on a very short store operation (PR_{VS})	0.65
Restore time (T_R)	50ns
Verify time (T_V)	50ns
Store current on the MTJ state transition from the Parallel (P) state to the Anti-Parallel (AP) state ($I_{P\rightarrow AP}$)	180μA
Store current on the MTJ state transition from the Anti-Parallel (AP) state to the Parallel (P) state ($I_{AP\rightarrow P}$)	102μA
Restore energy (E_R)	2.23pJ
Verify energy (E_V)	2.23pJ
Voltage (V)	1.2V

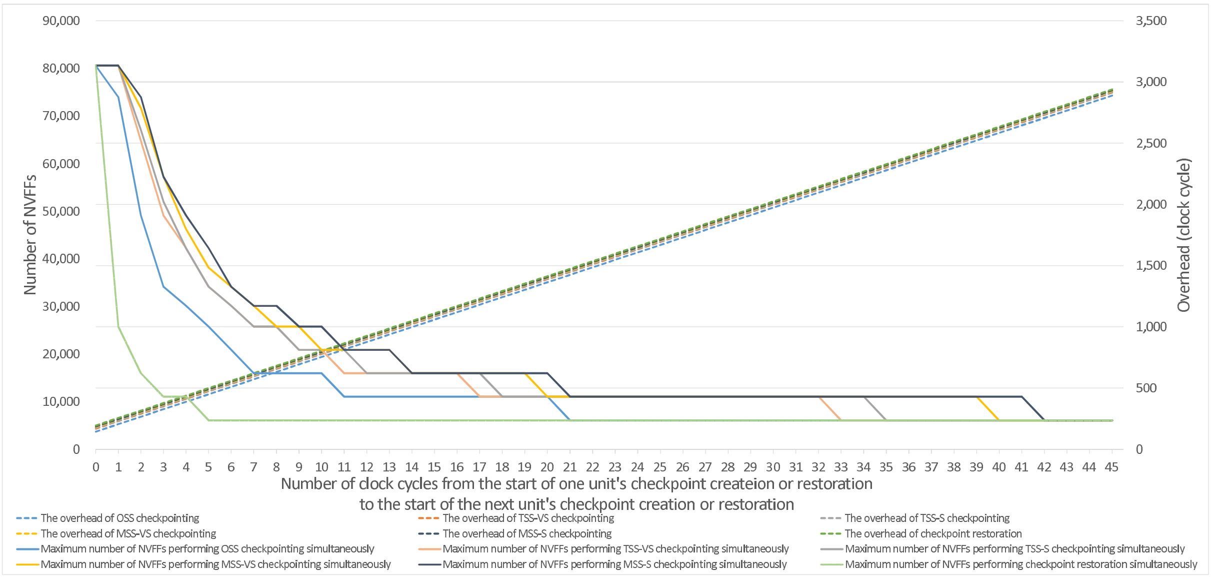


Fig. 11 This figure shows the trade-offs with the unit-interleave function on OSS checkpointing, TSS-VS checkpointing, TSS-S checkpointing, MSS-VS checkpointing, MSS-S checkpointing, and checkpoint restoration.

operation is a restore operation that restores the MTJ's value to the verification circuit, we assume the energy consumption and time for the verify operation in Table 3 are the same as for the restore operation.

5.2 The Trade-off between the Overhead and the Peak Current on Checkpoint Creations and Restorations

On checkpoint creations and restorations, the more NVFFs stored simultaneously, the smaller the overhead and the larger the peak current. Since the NVIOC should support various applications, we must consider cases where we want to reduce overhead while increasing peak current and cases where we want to reduce peak current while increasing overhead. Therefore, We designed the unit-interleave function to control the trade-off between the overhead and the peak current on checkpoint creations and restorations. We design the function based on the fact that the smaller the number of NVFFs performing checkpoint creations or restorations simultaneously, the smaller the peak current. Precisely, the function controls the number of clock cycles from the start of one unit's checkpoint creation or restoration to the start of the next unit's checkpoint creation or restoration, respectively. We evaluate the trade-off with the unit-interleave function on OSS checkpointing, TSS-VS checkpointing, TSS-S checkpointing, MSS-VS checkpointing, MSS-S checkpointing, and checkpoint restoration.

5.2.1 OSS Checkpointing

We evaluate the trade-off between the overhead and the peak current on OSS checkpointing. The time of the first store operation is set to T_L ($140ns$). The blue lines of Fig. 11 show the trade-off between the overhead (O_{OSS}) of OSS checkpointing and the maximum number (N_{OSS}) of NVFFs that perform checkpoint creations simultaneously when the number of clock cycles (C_{OSS}) from the start of one unit's checkpoint creation to the start of the next unit's checkpoint creation is varied. While O_{OSS} is always linearly proportional to C_{OSS}, N_{OSS} is inversely proportional to C_{OSS}. In the range, $0 <= C_{OSS} <= 7$, the value of N_{OSS} decreases rapidly as the value of C_{OSS} increases. In the range, $8 <= C_{OSS} <= 21$, the value of N_{OSS} decreases a little as the value of C_{OSS} increases. In the range, $22 <= C_{OSS}$, the value of N_{OSS} remains constant as the value of C_{OSS} increases. Therefore, even when N_{OSS} is minimized, O_{OSS} is only $14.25\mu sec$.

Note that even if C_{OSS} is 0, O_{OSS} takes $1.44\mu sec$. In case C_{OSS} is 0, O_{OSS} is qual to the OSS checkpointing overhead of a single unit since all units start checkpoint creations at a time.

5.2.2 TSS-VS Checkpointing

We evaluate the trade-off between the overhead and the peak current on TSS-VS checkpointing. The time of the first store operation is set to T_{VS} ($20ns$). The time of the second store operation is set to T_L ($140ns$). The orange lines of Fig. 11 show the trade-off between the overhead (O_{TSS-VS}) of TSS-VS checkpointing and the maximum number (N_{TSS-VS}) of NVFFs that perform checkpoint creations simultaneously when the number of clock cycles (C_{TSS-VS}) from the start of one unit's checkpoint creation to the start of the next unit's checkpoint creation is varied. While O_{TSS-VS} is always linearly proportional to C_{TSS-VS}, N_{TSS-VS} is inversely proportional to C_{TSS-VS}. In the range, $0 <= C_{TSS-VS} <= 11$, the value of N_{TSS-VS} decreases rapidly as the value of C_{TSS-VS} increases. In the range, $12 <= C_{TSS-VS} <= 33$, the value of

N_{TSS-VS} decreases a little as the value of C_{TSS-VS} increases. In the range, 34 <= C_{TSS-VS}, the value of N_{TSS-VS} remains constant as the value of C_{TSS-VS} increases. Therefore, even when N_{TSS-VS} is minimized, O_{TSS-VS} is only 21.81μsec.

Note that even if C_{TSS-VS} is 0, O_{TSS-VS} takes 1.68μsec. In case C_{TSS-VS} is 0, O_{TSS-VS} is equal to the TSS-VS checkpointing overhead of a single unit since all units start checkpoint creations at a time.

5.2.3 TSS-S Checkpointing

We evaluate the trade-off between the overhead and the peak current on TSS-S checkpointing. The time of the first store operation is set to T_S (40ns). The time of the second store operation is set to T_L (140ns). The gray lines of Fig. 11 show the trade-off between the overhead (O_{TSS-S}) of TSS-S checkpointing and the maximum number (N_{TSS-S}) of NVFFs that perform checkpoint creations simultaneously when the number of clock cycles (C_{TSS-S}) from the start of one unit's checkpoint creation to the start of the next unit's checkpoint creation is varied. While O_{TSS-S} is always linearly proportional to C_{TSS-S}, N_{TSS-S} is inversely proportional to C_{TSS-S}. In the range, 0 <= C_{TSS-S} <= 12, the value of N_{TSS-S} decreases rapidly as the value of C_{TSS-S} increases. In the range, 13 <= C_{TSS-S} <= 35, the value of N_{TSS-S} decreases a little as the value of C_{TSS-S} increases. In the range, 36 <= C_{TSS-S}, the value of N_{TSS-S} remains constant as the value of C_{TSS-S} increases. Therefore, even when N_{TSS-S} is minimized, O_{TSS-S} is only 23.07μsec.

Note that even if C_{TSS-S} is 0, O_{TSS-S} takes 1.72μsec. In case C_{TSS-S} is 0, O_{TSS-S} is equal to the TSS-S checkpointing overhead of a single unit since all units start checkpoint creations at a time.

5.2.4 MSS-VS Checkpointing

We evaluate the trade-off between the overhead and the peak current on MSS-VS checkpointing. The time of the first store operation is set to T_{VS} (20ns). The time of the second store operation is set to T_L (140ns). The yellow lines of Fig. 11 show the trade-off between the overhead (O_{MSS-VS}) of MSS-VS checkpointing and the maximum number (N_{MSS-VS}) of NVFFs that perform checkpoint creations simultaneously when the number of clock cycles (C_{MSS-VS}) from the start of one unit's checkpoint creation to the start of the next unit's checkpoint creation is varied. While O_{MSS-VS} is always linearly proportional to C_{MSS-VS}, N_{MSS-VS} is inversely proportional to C_{MSS-VS}. In the range, 0 <= C_{MSS-VS} <= 14, the value of N_{MSS-VS} decreases rapidly as the value of C_{MSS-VS} increases. In the range, 15 <= C_{MSS-VS} <= 40, the value of N_{MSS-VS} decreases a little as the value of C_{MSS-VS} increases. In the range, 41 <= C_{MSS-VS}, the value of N_{MSS-VS} remains constant as the value of C_{MSS-VS} increases. Therefore, even when N_{MSS-VS} is minimized, O_{MSS-VS} is only 26.22μsec.

Note that even if C_{MSS-VS} is 0, O_{MSS-VS} takes 1.82μsec. In case C_{MSS-VS} is 0, O_{MSS-VS} is equal to the MSS-VS checkpointing overhead of a single unit since all units start checkpoint creations at a time.

5.2.5 MSS-S Checkpointing

We evaluate the trade-off between the overhead and the peak current on MSS-S checkpointing. The time of the first store operation is set to T_S (40ns). The time of the second store operation is set to T_L (140ns). The black lines of Fig. 11 show the trade-off between the overhead (O_{MSS-S}) of MSS-S checkpointing and the maximum number (N_{MSS-S}) of NVFFs that perform checkpoint creations simultaneously when the number of clock cycles (C_{MSS-S}) from the start of one unit's checkpoint creation to the start of the next unit's checkpoint creation is varied. While O_{MSS-S} is always linearly proportional to C_{MSS-S}, N_{MSS-S} is inversely proportional to C_{MSS-S}. In the range, 0 <= C_{MSS-S} <= 14, the value of N_{MSS-S} decreases rapidly as the value of C_{MSS-S} increases. In the range, 15 <= C_{MSS-S} <= 42, the value of N_{MSS-S} decreases a little as the value of C_{MSS-S} increases. In the range, 43 <= C_{MSS-S}, the value of N_{MSS-S} remains constant as the value of C_{MSS-S} increases. Therefore, even when N_{MSS-S} is minimized, O_{MSS-S} is only 27.48μsec.

Note that even if C_{MSS-S} is 0, O_{MSS-S} takes 1.86μsec. In case C_{MSS-S} is 0, O_{MSS-S} is equal to the MSS-S checkpointing overhead of a single unit since all units start checkpoint creations at a time.

5.2.6 Checkpoint Restoration

We evaluate the trade-off between the overhead and the peak current on checkpoint restorations. The time of a restore operation is set to T_R (50ns). The green lines of Fig. 11 show the trade-off between the overhead (O_R) of checkpoint restorations and the maximum number (N_R) of NVFFs that perform checkpoint restorations simultaneously when the number of clock cycles (C_R) from the start of one unit's checkpoint restoration to the start of the next unit's checkpoint restoration is varied. While O_R is always linearly proportional to C_R, N_R is inversely proportional to C_R. In the range, 0 <= C_R <= 5, the value of N_R decreases rapidly as the value of C_R increases. In the range, 6 <= C_R, the value of N_R remains constant as the value of C_R increases. Therefore, even when N_R is minimized, O_R is only 4.99μsec.

Note that even if C_R is 0, O_R takes 1.94μsec. In case C_R is 0, O_R is equal to the checkpoint restoration overhead of a single unit since all units start checkpoint restorations at a time.

5.3 The IPC of Benchmark Executions with Checkpoint Creations

We evaluated the overhead of checkpoint creations by running benchmarks with periodic checkpoint creations and measuring each benchmark's IPC (Instruction Per Clock cycle). We used adpcm, bitcount, qsort, string search, sha, and

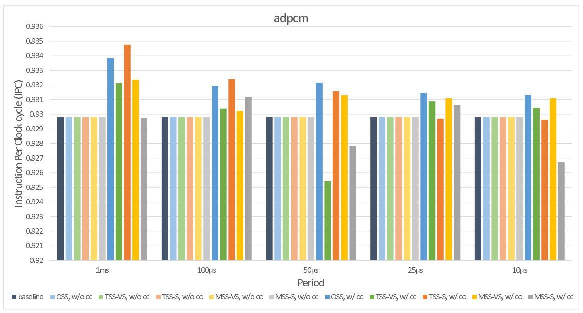


Fig. 12 This figure shows the IPC of the adpcm benchmark execution with periodic checkpoint creations.

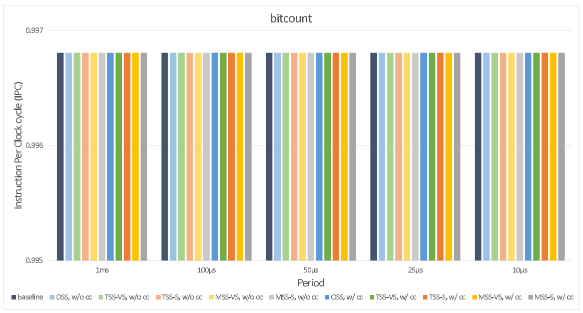


Fig. 13 This figure shows the IPC of the bitcount benchmark execution with periodic checkpoint creations.

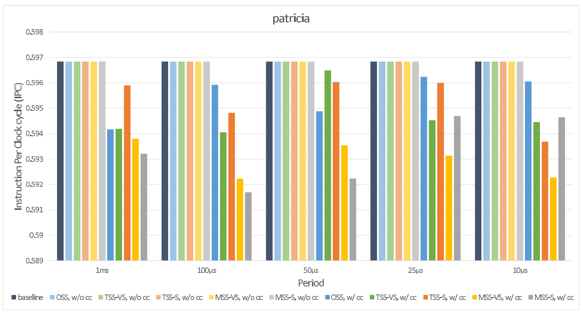


Fig. 14 This figure shows the IPC of the patricia benchmark execution with periodic checkpoint creation.

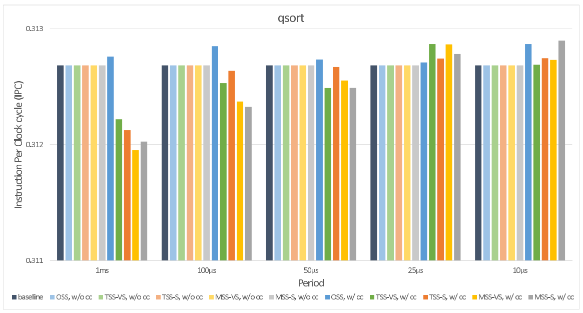


Fig. 15 This figure shows the IPC of the qsort benchmark execution with periodic checkpoint creation.

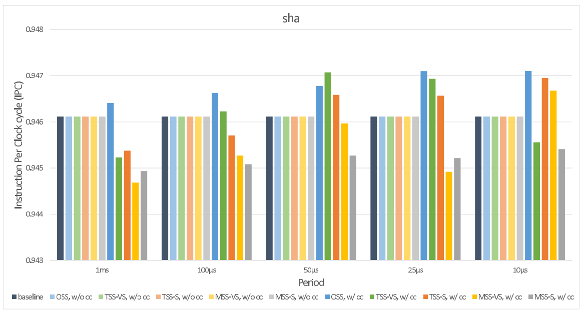


Fig. 16 This figure shows the IPC of the sha benchmark execution with periodic checkpoint creations.

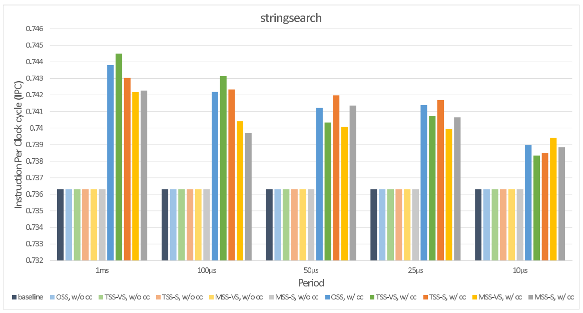


Fig. 17 This figure shows the IPC of the stringsearch benchmark execution with periodic checkpoint creation.

patricia from Mibench [37], a benchmark for embedded systems. Since we assume the case of minimizing peak current, the value of the unit-interleaving function is set to 21 clock cycles for OSS checkpointing, 33 clock cycles for TSS-VS checkpointing, 35 clock cycles for TSS-S checkpointing, 40 clock cycles for MSS-VS checkpointing, and 42 clock cycles for MSS-S checkpointing. Therefore, the OSS checkpointing overhead is 14.25μs, the TSS-VS checkpointing overhead is 21.81μs, the TSS-S checkpointing overhead is 23.07μs, the MSS-VS checkpointing overhead is 26.22μs, and the MSS-S checkpointing overhead is 27.48μs based on Fig. 11.

Figure 12, Fig. 13, Fig. 14, Fig. 15, Fig. 16, and Fig. 17 show the IPC of each benchmark execution with periodic checkpoint creations. For the baseline evaluation, we executed the benchmark without a checkpoint creation. For the other evaluations, we executed benchmarks while performing periodic checkpoint creations with various periods and various checkpoint creation methods. The NVDCs perform periodic checkpoint creations using the auto-store function. We discuss the evaluations without consistency checks (w/o cc) and the evaluations with consistency checks (w/ cc).

5.3.1 Without Consistency Checks

We focus on the evaluations which do not perform consistency checks. Although the data cache can check data consistency using the stored bit, we disable this function to evaluate the IPC when only periodic checkpoint creations are performed. The evaluations show that each benchmark execution achieves the same IPC as the baseline IPC, even with any checkpoint creation period and any checkpoint creation method. Since each checkpoint creation method has a different overhead, the evaluations also show that each benchmark execution achieves the same performance as the baseline, even with any checkpoint creation overhead. These results show that the NVFFM can hide the checkpoint creation overhead because it has two banks, and while one bank performs a checkpoint creation, the other bank operates as a simple register file.

5.3.2 With Consistency Checks

We focus on the evaluations which perform consistency checks. The data cache automatically checks data consistency and creates a checkpoint when it detects that a consistency-aware checkpoint creation is required. Figure 13 shows that the bitcount benchmark achieves the same IPC as the baseline IPC for any checkpointing period and any checkpointing method. The bitcount benchmark generates very little write-back. Therefore, the additional condition of write-back did not affect IPC for any configurations. In contrast, the other figures show that the other benchmarks achieve an IPC slightly different from the baseline IPC for each checkpointing period and each checkpointing method. These benchmarks generate many write-back. Therefore, the additional condition of write-back had some effect on IPC for each configuration.

The IPC difference was caused by allowing write-back only when the stored bit is 1 for data consistency checks. If a cache line with a stored bit of 0 requests a write-back, it will wait in the write buffer until a checkpoint creation is complete and the stored bit becomes 1. Therefore, data consistency checks may reduce the frequency of writes from the write buffer to the MRAM. In this case, performance may be improved because the instruction cache is more likely to acquire a bus grant. Data consistency checks may also cause more entries to accumulate in the write buffer. The CPU stalls if the write buffer is full, resulting in performance degradation. In either case, it is undesirable for data consistency checks to vary performance because WCET estimation becomes difficult. Furthermore, a consistency-aware checkpoint creation occurs at times unintended by the user to maintain data consistency, which may lead to unintended behavior on a checkpoint restoration. To solve these problems, we should introduce a mechanism like COACH [34] that guarantees data consistency without a consistency-aware checkpoint creation in future work.

5.4 The Energy Consumption of Checkpoint Creations

We evaluate the energy consumption of checkpoint creation with *SO*, *OSS*, *TSS*, and *MSS* methodologies. We calculate the energy consumption of checkpoint creation by considering only the store operation's energy consumption and the verify operation's energy consumption. We assume N_S is the total number of NVFFs that should perform checkpoint creation, N_{1S} is the number of NVFFs such that the FF's value is different from the MTJ's value after the first verify operation, and N_{2S} is the number of NVFFs such that the FF's latch value is different from the MTJ's value after the second verify operation. Other parameters are defined in Table 3. In case a VR-NVFF performs a store operation, the state of one MTJ transitions to the Anti-Parallel (AP) state in the first half-time of the store operation, then the state of the other MTJ transitions to the Parallel (P) state in the second half-time of the store operation [15], [16]. Regardless of the FF's value, the current ($I_{P\to AP}$) flows in one MTJ during the first half of the store operation, then the current ($I_{AP\to P}$) flows in the other MTJ during the second half of the store operation. In the case of *SO* method, the energy consumption (E_{so}) is calculated as:

$$E_{SO-store} = N_S * V * \frac{T_L}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{1}$$

In the case of *OSS* method, the energy consumption (E_{OSS}) is calculated as:

$$E_{OSS-verify} = N_S * E_V \tag{2}$$

$$E_{OSS-store} = N_{1S} * V * \frac{T_L}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{3}$$

$$E_{OSS} = E_{OSS-verify} + E_{OSS-store} \tag{4}$$

In the case of *TSS* method, the energy consumption (E_{TSS}) is calculated as:

$$E_{TSS-1st-verify} = N_S * E_V \tag{5}$$

$$E_{TSS-1st-store} = N_{1S} * V * \frac{T_{1st-store}}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{6}$$

$$N_{2S} = N_{1S} * (1 - PR_S) \tag{7}$$

$$E_{TSS-2nd-store} = N_{2S} * V * \frac{T_{2nd-store}}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{8}$$

$$\begin{aligned} E_{TSS} = {} & E_{TSS-1st-verify} + E_{TSS-1st-store} \\ & + E_{TSS-2nd-verify} + E_{TSS-2nd-store} \end{aligned} \tag{9}$$

In the case of *MSS* method such that the user-defined maximum value is set to zero, the energy consumption (E_{MSS}) is calculated as:

$$E_{MSS-1st-verify} = N_S * E_V \tag{10}$$

$$E_{MSS-1st-store} = N_{1S} * V * \frac{T_{1st-store}}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{11}$$

$$E_{MSS-2nd-verify} = N_{1S} * E_V \tag{12}$$

$$N_{2S} = N_{1S} * (1 - PR_S) \tag{13}$$

$$E_{MSS-2nd-store} = N_{2S} * V * \frac{T_{2nd-store}}{2} * (I_{AP\to P} + I_{P\to AP}) \tag{14}$$

$$E_{MSS-3rd-verify} = N_{2S} * E_V \tag{15}$$

$$\begin{aligned} E_{MSS} = {} & E_{MSS-1st-verify} + E_{MSS-1st-store} \\ & + E_{MSS-2nd-verify} + E_{MSS-2nd-store} \\ & + E_{MSS-3rd-verify} \end{aligned} \tag{16}$$

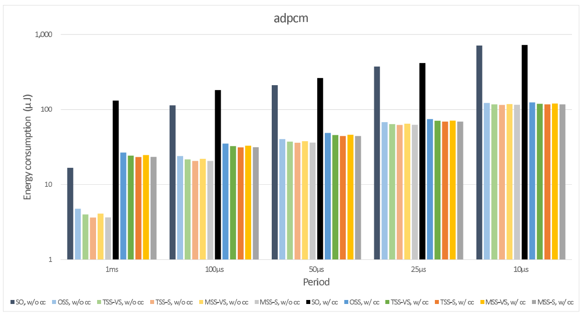


Fig. 18 This figure shows the energy consumption of checkpoint creations on the adpcm benchmark execution. The vertical axis is a logarithmic scale.

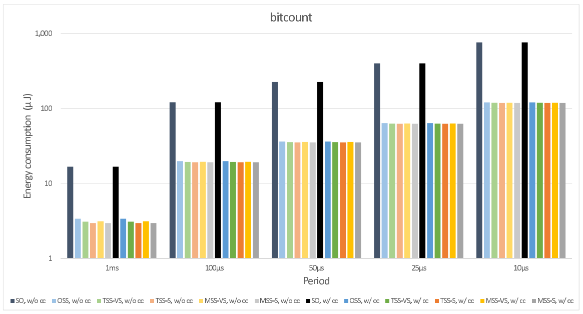


Fig. 19 This figure shows the energy consumption of checkpoint creations on the bitcount benchmark execution. The vertical axis is a logarithmic scale.

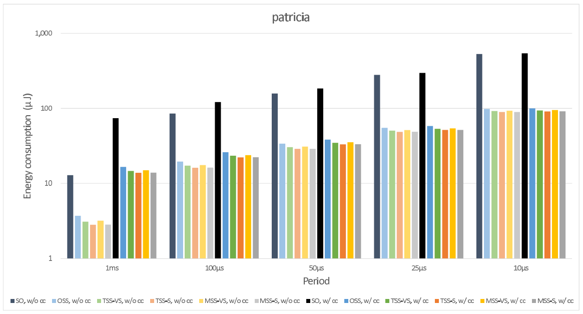


Fig. 20 This figure shows the energy consumption of checkpoint creations on the patricia benchmark execution. The vertical axis is a logarithmic scale.

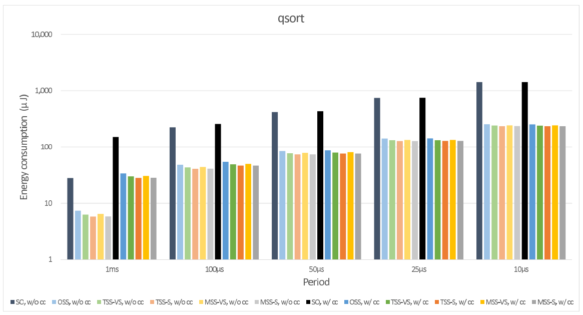


Fig. 21 This figure shows the energy consumption of checkpoint creations on the qsort benchmark execution. The vertical axis is a logarithmic scale.

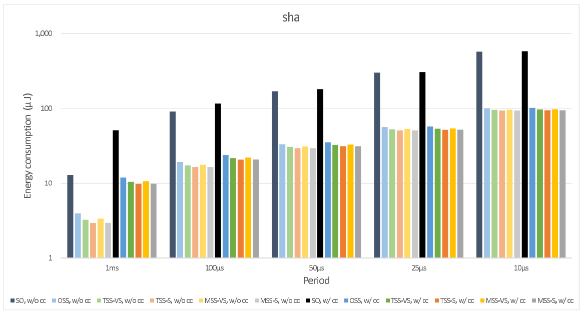


Fig. 22 This figure shows the energy consumption of checkpoint creations on the sha benchmark execution. The vertical axis is a logarithmic scale.

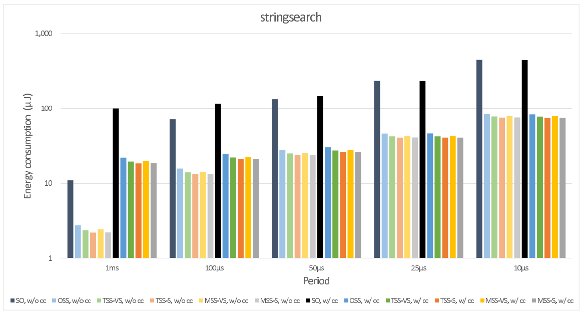


Fig. 23 This figure shows the energy consumption of checkpoint creations on the stringsearch benchmark execution. The vertical axis is a logarithmic scale.

The energy consumption of TSS-S checkpointing (E_{TSS-S}) is E_{TSS} calculated by substituting T_S for $T_{1st-store}$ and T_L for $T_{2nd-store}$. Similarly, the energy consumption of TSS-VS checkpointing (E_{TSS-VS}) is E_{TSS} calculated by substituting T_{VS} for $T_{1st-store}$ and T_L for $T_{2nd-store}$. The energy consumption of MSS-S checkpointing (E_{MSS-S}) is E_{MSS} calculated by substituting T_S for $T_{1st-store}$ and T_L for $T_{2nd-store}$. Similarly, the energy consumption of MSS-VS checkpointing (E_{MSS-VS}) is E_{MSS} calculated by substituting T_{VS} for $T_{1st-store}$ and T_L for $T_{2nd-store}$. We execute the benchmarks with periodic checkpoint creations, measure N_S and N_{1S} for each checkpoint creation, and calculate total energy consumption.

Figure 18, Fig. 19, Fig. 20, Fig. 21, Fig. 22, and Fig. 23 show the energy consumption of checkpoint creations on each benchmark execution with periodic checkpoint creations. For the baseline evaluation, we executed the benchmark without a checkpoint creation. For the other evaluations, we executed benchmarks while performing periodic checkpoint creations with various periods and various checkpoint creation methods. The NVDCs perform periodic checkpoint creations using the auto-store function. We discuss the evaluations without consistency checks (w/o cc) and the evaluations with consistency checks (w/ cc).

5.4.1 Without Consistency Checks

We focus on the evaluations which do not perform consistency checks. Although the data cache can check data consistency using a stored bit, we disable this function to evaluate the energy consumption of checkpoint creations when only periodic checkpoint creations are performed.

The evaluation shows that the shorter the cycle, the greater the energy consumption in any checkpoint method. SO checkpointing performs store operation on all NVFFs. Therefore power consumption of SO checkpointing increases significantly as the number of checkpoints increases. OSS checkpointing significatly reduces energy consumption compared to SO checkpointing in any period. This is because OSS checkpointing performs a verify operation before performing a store operation so that only NVFFs for which the MTJ value differs from the FF value are subject to the store operation. TSS checkpointing and MSS checkpointing reduce energy consumption compared to OSS checkpointing in any period. They can effectively reduce energy consumption due to lower the time of the first store operation. Since MSS checkpointing performs a third verify operation, E_{MSS-S} is slightly larger than E_{TSS-S}, and E_{MSS-VS} is also slightly larger than E_{TSS-VS}. In the case of MSS-VS checkpointing or TSS-VS checkpointing, the first store operation time is very short. Therefore, in the first store, the energy consumption decreases while the pass rate decreases. Because of the low pass rate, the number of NVFFs required to perform the second store operation are increased. As a result, the total energy consumption of MSS-VS checkpointing or TSS-VS checkpointing is more significant than that of MSS-S checkpointing or TSS-S checkpointing. We can reduce the total energy consumption by determining the first store operation time, considering the trade-off between the energy consumption of the first store operation and the pass rate of the first store operation.

5.4.2 With Consistency Checks

We focus on the evaluations which perform consistency checks. The data cache automatically checks data consistency and creates a checkpoint when it detects that a consistency-aware checkpoint creation is required.

The evaluations show that as the period increases, the energy consumption is greater than the energy consumption for the case without consistency checks. The longer the period, the greater the number of cache lines whose stored bit is zero. The NVIOC performs a consistency-aware checkpoint creation if a cache line whose stored bit is 0 requests a write-back. Therefore, the longer the period, the greater the number of consistency-aware checkpoint creations, resulting in increased energy consumption. On the one hand, ALU-intensive benchmark such as bitcount generates few write-back. Therefore, it causes few consistency-aware checkpoint creations, resulting in almost the same energy consumption as for the case without consistency checks. On the other hand, Memory-intensive benchmarks such as qsort generate many write-backs. Therefore, they cause many consistency-aware checkpoint creations, resulting in higher energy consumption than the case without consistency checks.

Table 4 The energy consumption of a checkpoint restoration

Restore energy	2.23pJ
N_R	80,598
E_R	179,733.54pJ

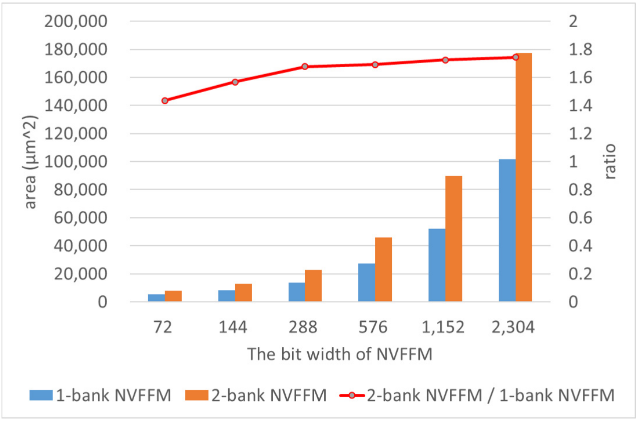


Fig. 24 This evaluation compares the area of a one-bank NVFFM with the area of a two-bank NVFFM while changing the bit width of the NVFFMs.

5.5 The Energy Consumption of a Checkpoint Restoration

In a checkpoint restoration, the valid bit is restored to select a bank with a valid checkpoint, and then the restore operation is performed. Therefore, the energy consumption of a checkpoint restoration (E_R) is calculated as $E_R = N_R * restore\_energy$ using the number of NVFFs performing the restore operation (N_R) and restore energy. The calculation results are shown in Table 4. From this calculation result, the energy consumption for a checkpoint restoration is only about 180nJ.

5.6 Area Overhead

We compared the area of a 1-bank NVFFM with the area of a 2-bank NVFFM while changing the bit width of the NVFFMs. The number of NVFF units in each bank is set to one. Therefore, the bit width of the NVFF unit is the same as that of the NVFFM. We used Synopsys Design Compiler to perform logic synthesis and compare the netlist area.

Figure 24 shows the evaluation results. The smaller the bit width, the smaller the ratio of 2-bank NVFF to 1-bank NVFF. This is because the circuit overhead, independent of the number of banks, is relatively large when the bit width is small. Therefore, when creating an NVFFM with a certain bit width (e.g., 8,192 bits wide), the total area will be smaller if the number of NVFF bits per unit is as large as possible. In addition, as the number of NVFF bits per unit increases, the number of NVFFs that simultaneously create and restore checkpoints increases, leading to an increase in peak current and a decrease in the overhead of checkpoint creations and restorations. Thus, NVFFM parameters should be determined by considering various trade-offs, such as the area overhead, the peak current, and the checkpoint creation and restoration overhead.

The area overhead of the 2-bank NVFFM is significantly large. However, the 2-bank NVFFM is desirable for a fault-tolerant real-time system in order to hide the overhead of checkpoint creations and to perform a checkpoint restoration normally even if a power failure occurs at any time, even considering the increase in area. NV register file, which combines multiple NVFFs, and NV-SRAM [38] may be useful for area-reduced designs while maintaining a 2-bank configuration. In particular, since the NVIOC uses many NVFFs for the data cache and the instruction cache, using NV-SRAM for the data cache and the instruction cache may significantly reduce the area while maintaining a 2-bank configuration.

6. Conclusion and Future Work

We design a control architecture for various non-volatile devices to realize a non-stop microprocessor that can operate normally immediately after power is restored, even if the power is turned off at any time, with little performance degradation. We also design the non-stop microprocessor based on the control architecture for non-volatile devices. The microprocessor also has a flexible control mechanism for non-volatile devices that can accommodate various situations while considering trade-offs. In conclusion, the non-stop microprocessor with such capabilities will be an essential component of a fault-tolerant real-time system with high schedulability.

In future work, we will research to achieve a non-stop system by making all system components, such as memory interfaces, IOs, buses, and networks, non-volatile.

Acknowledgements

This work was supported by the following people:

Sony Semiconductor Solutions Corporation
Mr. Kazuhiro Bessho
Mr. Kenta Suzuki
Mr. Keizo Hiraga

This work was supported by JST SPRING, Grant Number JPMJSP2123.

References

[1] S. Punnekkat, A. Burns, and R. Davis, "Analysis of checkpointing for real-time systems," Real-Time Systems, vol.20, no.1, pp.83–102, Jan. 2001.

[2] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," IEEE Trans. Comput., vol.53, no.2, pp.217–231, 2004.

[3] A. Ejlali, B.M. Al-Hashimi, M.T. Schmitz, P. Rosinger, and S.G. Miremadi, "Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.14, no.4, pp.323–335, 2006.

[4] M. Salehi, M.K. Tavana, S. Rehman, M. Shafique, A. Ejlali, and J. Henkel, "Two-state checkpointing for energy-efficient fault tolerance in hard real-time systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.24, no.7, pp.2426–2437, 2016.

[5] S. Ikegawa, F.B. Mancoff, J. Janesky, and S. Aggarwal, "Magnetoresistive random access memory: Present and future," IEEE Trans. Electron Devices, vol.67, no.4, pp.1407–1419, 2020.

[6] T. Mikolajick, U. Schroeder, and S. Slesazeck, "The past, the present, and the future of ferroelectric memories," IEEE Trans. Electron Devices, vol.67, no.4, pp.1434–1443, 2020.

[7] Y. Chen, "Reram: History, status, and future," IEEE Trans. Electron Devices, vol.67, no.4, pp.1420–1433, 2020.

[8] T. Kim and S. Lee, "Evolution of phase-change memory for the storage-class memory and beyond," IEEE Trans. Electron Devices, vol.67, no.4, pp.1394–1406, 2020.

[9] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai, "Nonvolatile magnetic flip-flop for standby-power-free socs," 2008 IEEE Custom Integrated Circuits Conference, pp.355–358, 2008.

[10] J. Wang, Y. Liu, H. Yang, and H. Wang, "A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications," The 2010 International Conference on Green Circuits and Systems, pp.646–650, 2010.

[11] M. Qazi, A. Amerasekera, and A.P. Chandrakasan, "A 3.4-pj feram-enabled d flip-flop in 0.13-μm cmos for nonvolatile processing in digital systems," IEEE J. Solid-State Circuits, vol.49, no.1, pp.202–211, 2014.

[12] K. Ali, F. Li, S.Y.H. Lua, and C.-H. Heng, "Energy- and area-efficient spin-orbit torque nonvolatile flip-flop for power gating architecture," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.26, no.4, pp.630–638, 2018.

[13] M. Kudo and K. Usami, "Nonvolatile power gating with mtj based nonvolatile flip-flops for a microprocessor," 2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp.1–6, 2017.

[14] M. Kudo, "Low Power technology of LSI by Fine-Grain Power Gating and Magnetic Tunnel Junction (in Japanese)," Ph.D. thesis, 2016.

[15] K. Usami, J. Akaike, S. Akiba, M. Kudo, H. Amano, T. Ikezoe, K. Hiraga, Y. Shuto, and K. Yagami, "Energy efficient write verify and retry scheme for mtj based flip-flop and application," 2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp.91–98, 2018.

[16] J.-G.(J.) Zhu and C. Park, "Magnetic tunnel junctions," Materials Today, vol.9, no.11, pp.36–45, 2006.

[17] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, J. Shu, and H. Yang, "Ambient energy harvesting nonvolatile processors: From circuit to system," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp.1–6, 2015.

[18] S. Ahmed, N.A. Bhatti, M. Brachmann, and M.H. Alizai, "A survey on program-state retention for transiently-powered systems," Journal of Systems Architecture, vol.115, p.102013, 2021.

[19] K. Ma, X. Li, S. Li, Y. Liu, J.J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile processor architecture exploration for energy-harvesting applications," IEEE Micro, vol.35, no.5, pp.32–40, 2015.

[20] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," SIGARCH Comput. Archit. News, vol.39, no.1, pp.159–170, March 2011.

[21] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, pp.330–335, 2014.

[22] Texas instruments, "MSP430FRxx microcontrollers," 2018.

[23] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B.O. Eversmann, "An 82 μa/mhz microcontroller with embedded

feram for energy-harvesting applications," 2011 IEEE International Solid-State Circuits Conference, pp.334–336, 2011.
[24] C. Pan, M. Xie, Y. Liu, Y. Wang, C.J. Xue, Y. Wang, Y. Chen, and J. Hu, "A lightweight progress maximization scheduler for non-volatile processor under unstable energy harvesting," vol.52, no.5, pp.101–110, June 2017.
[25] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," 2012 Proceedings of the ESSCIRC (ESSCIRC), pp.149–152, 2012.
[26] Y. Wang, Y. Liu, S. Li, X. Sheng, D. Zhang, M.-F. Chiang, B. Sai, X.S. Hu, and H. Yang, "Pacc: A parallel compare and compress codec for area reduction in nonvolatile processors," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.22, no.7, pp.1491–1505, 2014.
[27] A. Roohi and R.F. DeMara, "Nv-clustering: Normally-off computing using non-volatile datapaths," IEEE Trans. Comput., vol.67, no.7, pp.949–959, 2018.
[28] M. Xie, M. Zhao, C. Pan, Jingtong Hu, Y. Liu, and C.J. Xue, "Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor," 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp.1–6, 2015.
[29] M. Xie, C. Pan, M. Zhao, Y. Liu, C.J. Xue, and J. Hu, "Avoiding data inconsistency in energy harvesting powered embedded systems," ACM Trans. Des. Autom. Electron. Syst., vol.23, no.3, pp.1–25, March 2018.
[30] Q. Liu and C. Jung, "Lightweight hardware support for transparent consistency-aware checkpointing in intermittent energy-harvesting systems," 2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp.1–6, 2016.
[31] F. Li, K. Qiu, M. Zhao, J. Hu, Y. Liu, Y. Guan, and C.J. Xue, "Checkpointing-aware loop tiling for energy harvesting powered nonvolatile processors," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.38, no.1, pp.15–28, 2019.
[32] W.S. Lim, C.-H. Tu, C.-F. Wu, and Y.-H. Chang, "icheck: Progressive checkpointing for intermittent systems," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol.40, no.11, pp.2224–2236, 2021.
[33] A. Hoseinghorban, M. Abbasinia, and A. Ejlali, "Prowl: A cache replacement policy for consistency aware renewable powered devices," IEEE Trans. Emerg. Topics Comput., vol.10, no.1, pp.476–487, 2022.
[34] A. Hoseinghorban, A.M.H. Monazzah, M. Bazzaz, B. Safaei, and A. Ejlali, "Coach: Consistency aware check-pointing for nonvolatile processor in energy harvesting systems," IEEE Trans. Emerg. Topics Comput., vol.9, no.4, pp.2076–2088, 2021.
[35] S. Senni, L. Torres, G. Sassatelli, and A. Gamatie, "Non-volatile processor based on mram for ultra-low-power iot devices," J. Emerg. Technol. Comput. Syst., vol.13, no.2, pp.1–23, Dec. 2016.
[36] A. Kamei, T. Kojima, H. Amano, D. Yokoyama, H. Miyauchi, K. Usami, K. Hiraga, K. Suzuki, and K. Bessho, "Energy saving in a multi-context coarse grained reconfigurable array with non-volatile flip-flops," 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp.273–280, 2021.
[37] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. no.01EX538), pp.3–14, Dec. 2001.
[38] S. Yamamoto, Y. Shuto, and S. Sugahara, "Nonvolatile flip-flop using pseudo-spin-transistor architecture and its power-gating applications," 2012 International Semiconductor Conference Dresden–Grenoble (ISCDG), pp.17–20, 2012.

Appendix A: Abbreviations Summary

This paper uses many abbreviations. Therefore, we prepare Table A· 1 to show the abbreviations for clarity.

Table A· 1 Abbreviations table

Abbreviation	Full word
WCET	Worst-Case Execution Time
FF	Flip-Flop
NVM	Non-Volatile Memory
NVFF	Non-Volatile Flip-Flop
MTJ	Magnetic Tunneling Junction
NVP	Non-Volatile Processor
IOC	IO Core
NVIOC	Non-Volatile IO Core
SoC	System-on-Chip
SiP	System-in-Package
NVFFM	Non-Volatile Flip-Flop Module
NVDC	Non-Volatile Device Controller
ECC	Error Correction Code
SO	Store Only
OSS	One Step Store
TSS	Two Step Store
MSS	Multi Step Store
IRQ	Interrupt ReQuests
PR	Pass Rate
IPC	Instruction Per Clock cycle
w/o cc	Without Consistency Check
w/ cc	With Consistency Check
AP state	Anti-Parallel state
P state	Parallel state

Shota Nakabeppu received B.S. and M.S. degrees from Keio University, Japan, in 2018 and 2020. Since April 2020, he has been pursuing the Ph.D degree in Electrical Engineering at Keio University.

Nobuyuki Yamasaki received a Ph.D. in engineering from Keio University in 1996. He is a professor in the Department of Information and Computer Science at Keio University. His research interests include real-time processing and communication, parallel and distributed processing, computer architecture, operating systems, embedded systems, and SoC design. He is a member of IEICE, IPSJ, RSJ, and IEEE.