

Standard Operating Procedure

Document Number: PQSYS-001

Standard Operating Procedure (SOP): Priority Queue Buffer
Sorting and Consumption

Revision: 1.0

Last Updated: [DATE]

Contents

1. Introduction
 - 1.1 Purpose
 - 1.2 System Overview
 - 1.3 Regulatory Compliance
2. System Specifications
 - 2.1 Buffer and Process Definitions
 - 2.2 Flag and State Parameters
3. Operational Protocols
 - 3.1 Standard Operating Procedures
 - 3.1.1 Simulation Task
 - 3.1.2 Sorting Task
 - 3.1.3 Consumption Task
 - 3.2 Performance Optimization
 - 3.2.1 Concurrency Handling
 - 3.2.2 Data Flow Assurance
4. Emergency Operations
 - 4.1 Buffer Blockage Handling
 - 4.2 Recovery from Sorting Delays
5. Maintenance Requirements
 - 5.1 Data Integrity Validation
 - 5.2 Buffer Reinitialization

- 6. Quality Assurance
 - 6.1 Buffer State Inspection
 - 6.2 Sorting Algorithm Conformance
 - 7. Security Protocols
 - 7.1 Flag Consistency
 - 7.2 Process Isolation
 - 8. Environmental Considerations
 - 8.1 Simulation Randomness Variability
 - 8.2 Output Rate Regulation
 - 9. Training Requirements
 - 9.1 Operator Familiarity
 - 9.2 Debug and Visualization Tools
 - 10. Document Control
 - 10.1 Revision History
 - 10.2 Authorization
 - 11. Process Flows and State Transitions
 - 11.1 Simulation to Sorting Sequence
 - 11.2 Sorting Stages
 - 11.3 Consumption Read Logic
-

1. Introduction

1.1 Purpose

To manage a buffered data pipeline where input values are generated, sorted, and consumed in real-time, ensuring the queue remains ordered and efficient.

1.2 System Overview

- The buffer has 6 positions: 3 for input (`in_f[1..3]`), 3 for output (`out_l[1..3]`).

- There are three continuous processes: `simulator` , `sort` , and `sim_cons` .
- `simulator` writes random values into the buffer.
- `sort` rearranges values using a rotation-based sorting algorithm.
- `sim_cons` consumes the data once sorting is complete.

1.3 Regulatory Compliance

- Ensures no data loss or overwrite during concurrent operations.
- Follows queue integrity and non-overlapping access guidelines.

2. System Specifications

2.1 Buffer and Process Definitions

- Buffer Inputs: `in_f[1..3]` , initialized to 0
- Buffer Outputs: `out_l[1..3]` , initialized to 0
- Processes:
 - `simulator` : Writes values into `in_f[1]` when it's zero
 - `sort` : Sorts buffer using a 3-cell rotation logic
 - `sim_cons` : Reads from `out_l[1]` when value is available

2.2 Flag and State Parameters

- `sort_req` : Set when a new value is inserted
 - `sort_OK` : True if sorting is complete
 - `state` : Tracks internal task progress (e.g. `initial` , `cell1` , `ready`)
 - `tmp` , `reg` : Temporary variables for sorting
-

3. Operational Protocols

3.1 Standard Operating Procedures

3.1.1 Simulation Task

- Monitor `in_f[1]`
- If zero, write a random value (1-8)
- Set `sort_req = TRUE`
- Loop back to wait for next empty slot

3.1.2 Sorting Task

- Triggered when `in_f[1] != 0`
- Enters multiple phases:
 - `cell1 → cell2 → cell3 → ready`
 - Temporarily holds values in `tmp`
 - Compares and swaps using `reg`
- When finished, set `sort_OK = TRUE`
- Reset `sort_req = FALSE`

3.1.3 Consumption Task

- Monitor `out_l[1]`
- If value present and `sort_OK = TRUE`, consume it (set to 0)
- Resume wait until next data is sorted

3.2 Performance Optimization

3.2.1 Concurrency Handling

- Use fairness constraints to keep all tasks progressing
- Sorting and consuming never run on same cell simultaneously

3.2.2 Data Flow Assurance

- Each value inserted is guaranteed to be sorted and consumed
 - Avoid buffer overflow by checking `in_f[1]` availability
-

4. Emergency Operations

4.1 Buffer Blockage Handling

- If `in_f[1]` remains full, block simulator
- Unblock only after sort and consume complete

4.2 Recovery from Sorting Delays

- If `sort_OK = FALSE`, hold `sim_cons` in idle
 - Resume once sorter signals ready
-

5. Maintenance Requirements

5.1 Data Integrity Validation

- Periodically inspect all `in_f` and `out_l` cells
- Check for invalid (non-zero) values post-consumption

5.2 Buffer Reinitialization

- Reset buffer values to 0 on system reset or critical fault
-

6. Quality Assurance

6.1 Buffer State Inspection

- Verify that buffer holds consistent states during handoff
- Ensure `sort_OK` transitions are reliable

6.2 Sorting Algorithm Conformance

- Validate behavior against Hoeg-Møllergaard-Staunstrup rotation method
- Confirm 3-step move logic is applied consistently

7. Security Protocols

7.1 Flag Consistency

- Ensure `sort_req` only rises on insertion
- Block multiple raises before sorting completes

7.2 Process Isolation

- Tasks execute independently via distinct state machines
- Shared variables gated through synchronized conditions

8. Environmental Considerations

8.1 Simulation Randomness Variability

- Ensure range 1-8 is used uniformly
- Pseudorandom generator should be deterministic for test mode

8.2 Output Rate Regulation

- Consumption rate tied to sort completion
 - Avoid overconsumption by gating with `sort_OK`
-

9. Training Requirements

9.1 Operator Familiarity

- Understanding of flag logic (`sort_req` , `sort_OK`)
- Awareness of state machine flow per task

9.2 Debug and Visualization Tools

- Use simulation trace tools to observe state transitions
 - Employ buffer viewers to inspect cell values
-

10. Document Control

10.1 Revision History

- Rev 1.0 - Initial SOP based on formal p-queue specification

10.2 Authorization

- Approved by: Systems Design Lead
 - Reviewed by: Embedded QA Team
-

11. Process Flows and State Transitions

11.1 Simulation to Sorting Sequence

- Simulator checks `in_f[1]` → Writes value → Sets `sort_req`

- Sort begins when `in_f[1] != 0` and `sort_req = TRUE`

11.2 Sorting Stages

- `initial` → `cell1` → `cell1_p` → `cell1_m` → ... → `ready`
- Data values are rotated based on comparisons and reg logic

11.3 Consumption Read Logic

- Waits for `out_l[1] != 0` and `sort_OK = TRUE`
- Sets `out_l[1] = 0` to complete data consumption cycle