WORK FORM		
Document Title	Document Description	Version No.
	Software Detailed Design, Circular Buffer on RAM	**0**

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Detailed Design Document has been completed.*

Version	Date	Description	Author
1.0.0	01/10/2023	Initial document	

Table of Contents

List of Figures

List of Tables

1. PURPOSE

The purpose of this document is to provide the detailed design of the Circular Buffer, which is useful in storage management of RAM in memory space constrain MCU.

2. SCOPE

The scope of this document is limited to the typical specification and workflow of the Circular Buffer.

3. DEFINITIONS

Table 1. Definitions, Acronyms, and Abbreviations

Acronyms	Terms	Definitions
CB	Circular Buffer	A single fixed-size buffer with end-to-end connection. Also known as Ring Buffer, Cyclic Buffer.
FIFO	First In First Out	A form of data structure manipulation where the oldest entry (data unit) is the first to be processed.
MCU	Micro Controller Unit	A hardware unit contains CPU(s) along with memory and GPIO(s) often use in embedded applications.

4. DETAILED DESIGN

4.1. Overview

A Circular Buffer is a single string of data buffers where the **head** (first position) and **tail** (last position) are connected, allowing overwrite operation to carry on from the head when the buffer is full.

The Circular Buffer uses FIFO logic, when removing some values, the oldest available will be the first to be removed.

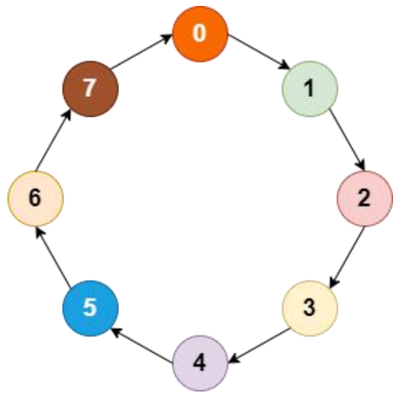


Figure 1. General Demonstration of Circular Buffer

The circular buffer is used in the following components:

- Command processor
- RAM logger
- Network manager
- Serial library

4.2. Reader and Writer

To control the flow of the read/write operation, Circular Buffer requires 2 pointers (or counter) known as **Reader** and **Writer**. Both will start at the head of the Circular Buffer, **Writer** move when the write operation occurs, and the **Reader** will advance when the read operation occurs.

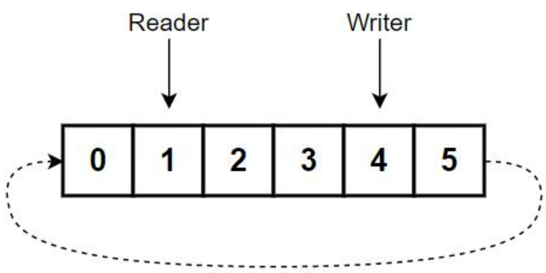


Figure 2. Circular Buffer Demonstration with Reader and Writer count

4.3. Full and Overflow

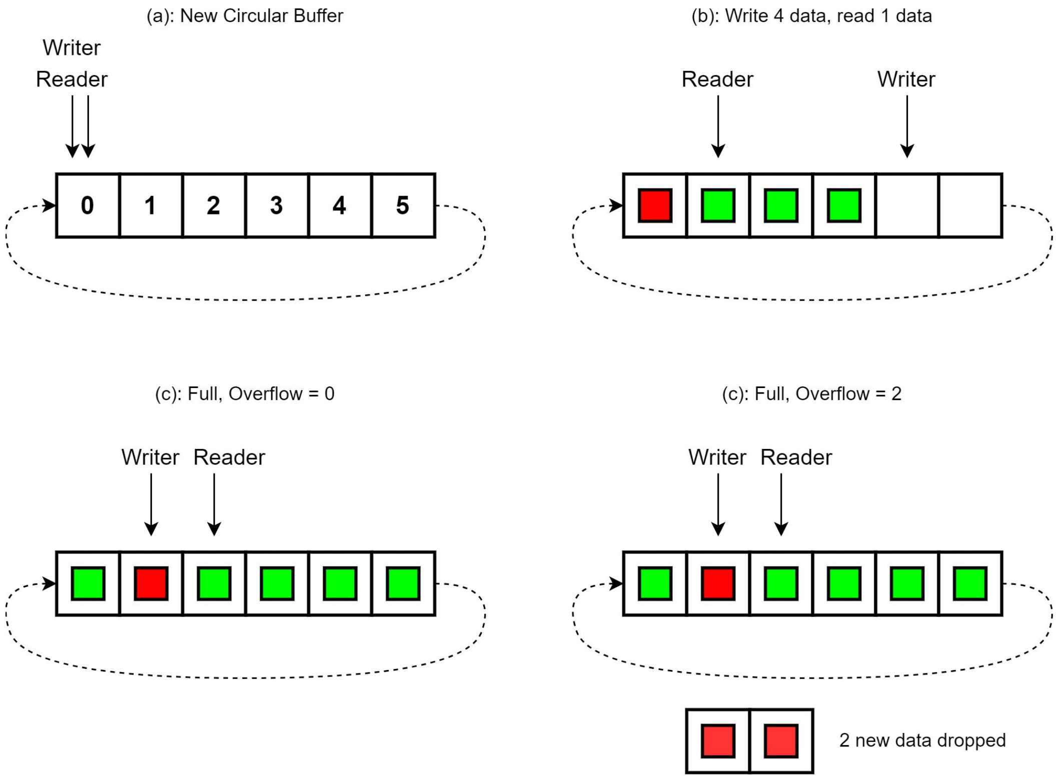


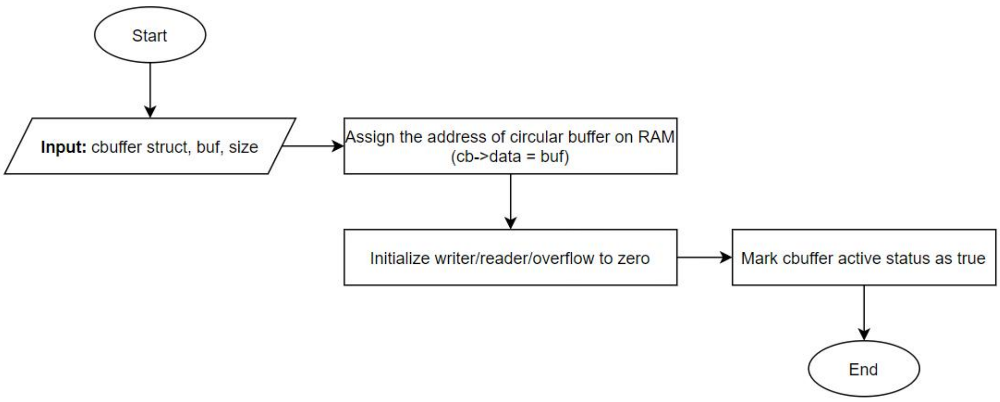
Figure 3. Demonstration of a full Circular Buffer

There are many ways to control the flow of a Circular Buffer, typically when a write operation takes place while the buffer is already full.

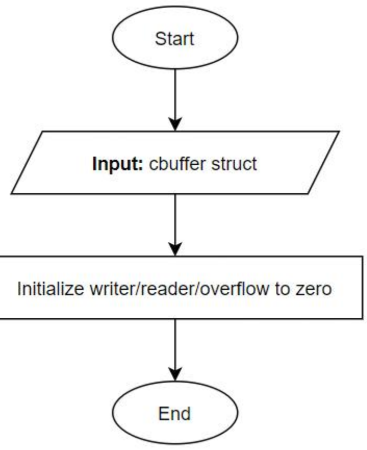
In the case of this document, the writer counter will no longer move when the buffer is full, a read operation is required to free up some space for incoming data, otherwise, the new data will be counted as lost, and will not be available in the Circular Buffer. Also, the buffer will have an "***Overflow***" variable which helps determine the number of lost data during this scenario.

5. API REFERENCES

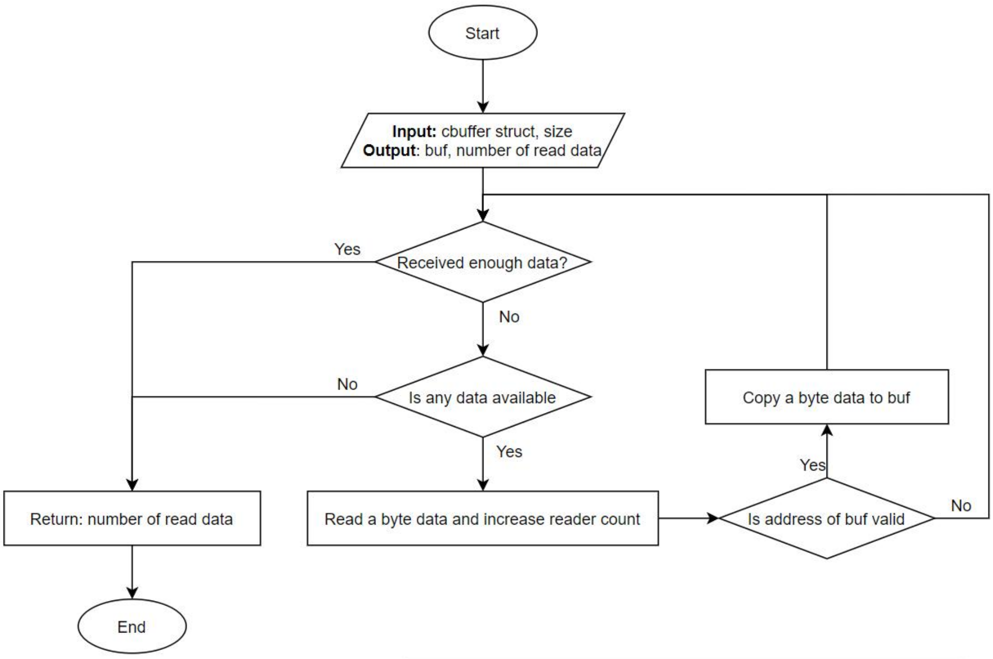
5.1. cb_init

API	void **cb_init**(cbuffer_t *cb, void *buf, uint32_t size)
Purpose	Initialize a new Circular Buffer
Description	**cb_init** requires input consisting of a circular buffer structure (cbuffer_t), pointer to a memory-allocated buffer (buf), and the size of the circular buffer (size). In the initialization process, the circular buffer will be marked as “active”, allowing the write operation to start at the first position of the buffer (head). **Note:** All Cbuffer must be initialized before any kind of action involved that certain buffer.
Flowchart	 Figure 4. Flowchart of cb_init API

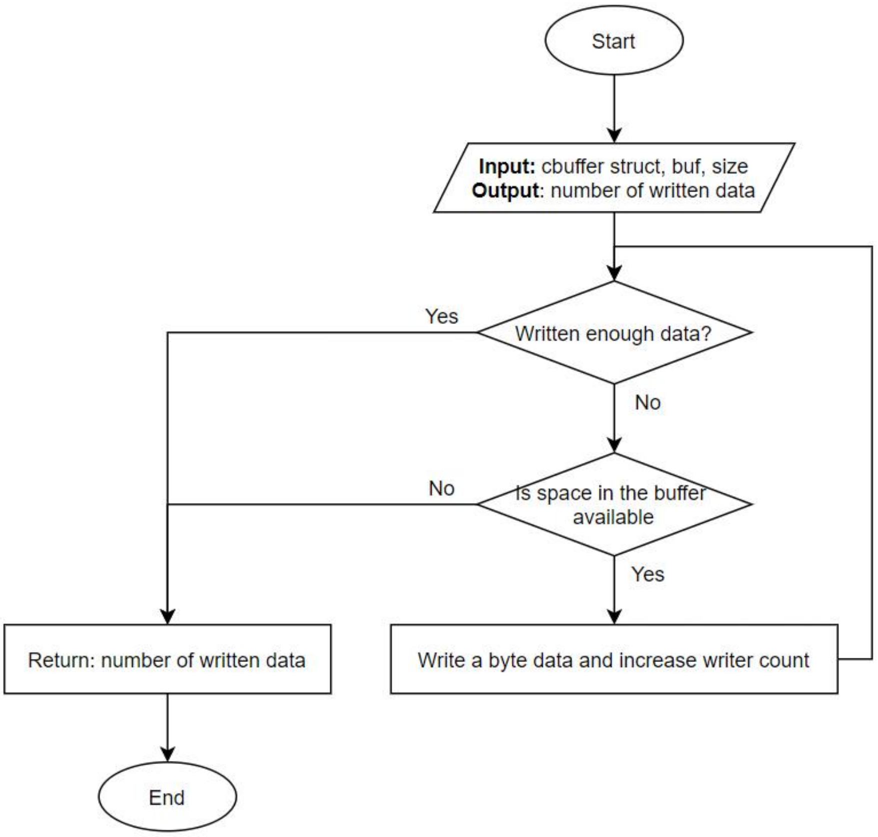
5.2. cb_clear

API	void **cb_clear**(cbuffer_t *cb)
Purpose	Clear counter and data of assigned Cbuffer.
Description	Reset Cbuffer’s reader, writer, and overflow counter, technically, ignore and allow overwrite of all currently valid data in Cbuffer.
Flowchart	 Figure 5. Flowchart of cb_clear API

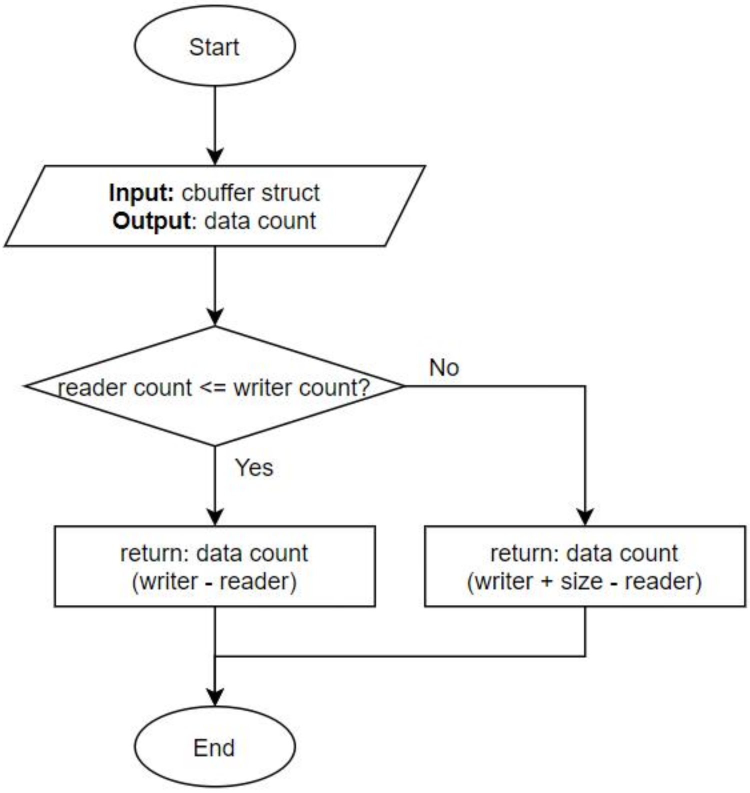
5.3. cb_read

API	uint32_t **cb_read**(cbuffer_t *cb, void *buf, uint32_t nbyte)
Purpose	Read a number of byte from Cbuffer and raise the reader count.
Description	Read “*n_byte”* number of data from Cbuffer and store it in “*buf” (*an external buffer). The actual read bytes will be returned.
Flowchart	 Figure 6. Flowchart of cb_read API

5.4. cb_write

API	uint32_t **cb_write**(cbuffer_t *cb, void *buf, uint32_t nbyte)
Purpose	Write a number of byte to Cbuffer and raise the writer count.
Description	Write "*n_byte*" number of data from "*buf*" and store it in cbuffer. The actual written bytes will be returned.
Flowchart	 Figure 7. Flowchart of cb_write API

5.5. cb_data_count

API	uint32_t **cb_data_count**(cbuffer_t *cb)
Purpose	Calculate readable bytes from cbuffer.
Description	**cb_data_count** calculate the number of readable bytes base on reader and writer count.
Flowchart	 Figure 8. Flowchart of cb_data_count API

5.6. cb_space_count

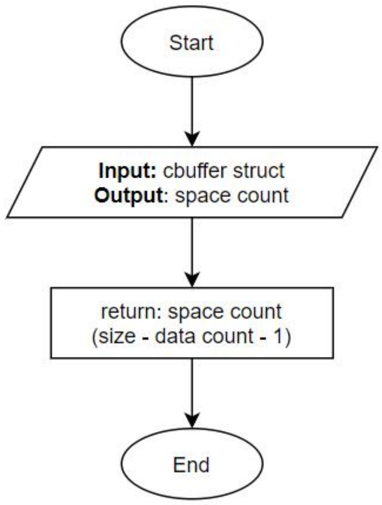
<table>
<tr><td>API</td><td>uint32_t cb_space_count(cbuffer_t *cb)</td></tr>
<tr><td>Purpose</td><td>Calculate the number of data spaces can be written to cbuffer.</td></tr>
<tr><td>Description</td><td>cb_space_count uses the size of cbuffer minus the number of readable bytes to determine the remaining space.</td></tr>
<tr><td>Flowchart</td><td>


Figure 9. Flowchart of cb_space_count API
</td></tr>
</table>