

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	

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

Table of Contents

1. PURPOSE .....4

2. SCOPE .....4

3. DEFINITIONS ..... 4

4. DETAILED DESIGN .....5

    4.1. Overview .....5

        The circular buffer is used in the following components: .....5

    4.2. Reader and Writer ..... 5

    4.3. Full and Overflow .....6

5. API REFERENCES .....7

    5.1. cb\_init ..... 7

    5.2. cb\_clear ..... 8

    5.3. cb\_read .....9

    5.4. cb\_write ..... 10

    5.5. cb\_data\_count..... 11

    5.6. cb\_space\_count ..... 12

<b>WORK FORM</b>		
Document Title	Document Description	Version No.
	<b>Software Detailed Design, Circular Buffer on RAM</b>	<b>0</b>

## List of Figures

Figure 1. General Demonstration of Circular Buffer .....	5
Figure 2. Circular Buffer Demonstration with Reader and Writer count .....	5
Figure 3. Demonstration of a full Circular Buffer .....	6
Figure 4. Flowchart of cb_init API .....	7
Figure 5. Flowchart of cb_clear API .....	8
Figure 6. Flowchart of cb_read API .....	9
Figure 7. Flowchart of cb_write API .....	10
Figure 8. Flowchart of cb_data_count API .....	11
Figure 9. Flowchart of cb_space_count API .....	12

## List of Tables

Table 1 . Definitions, Acronyms, and Abbreviations .....	4
--	---

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

**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.

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

## 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.

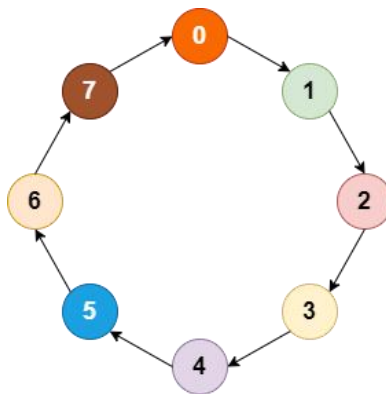


Figure 1. General Demonstration of Circular Buffer

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.

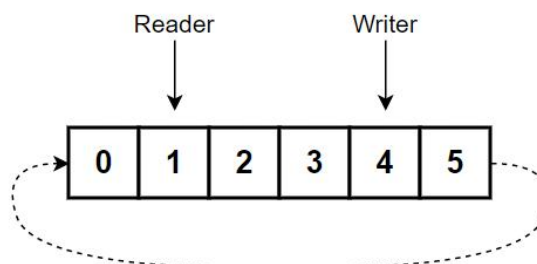


Figure 2. Circular Buffer Demonstration with Reader and Writer count

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

### 4.3. Full and Overflow

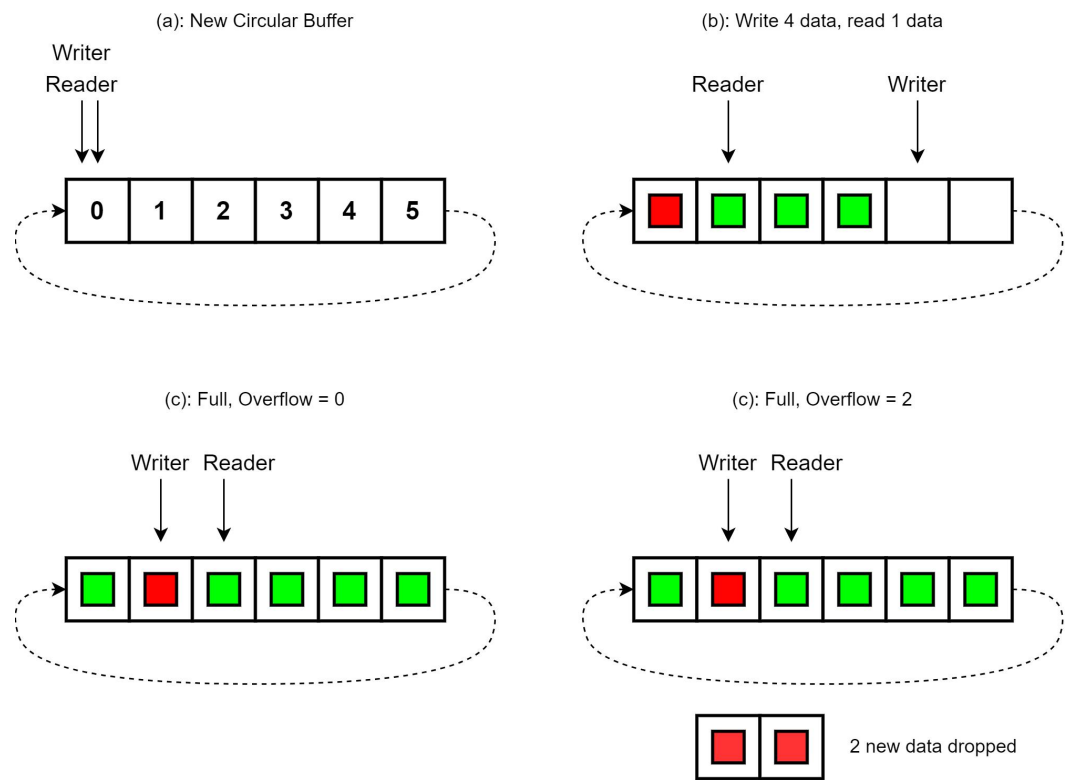


Figure 3. Demonstration of a full Circular Buffer

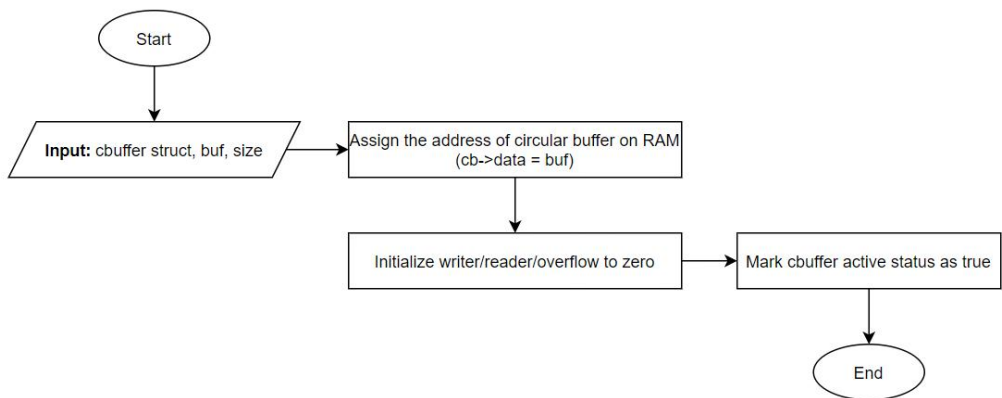
There are many ways to control the flow of a Circular Buffer, typically when a write operation takes place while the buffer is already full.

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.

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

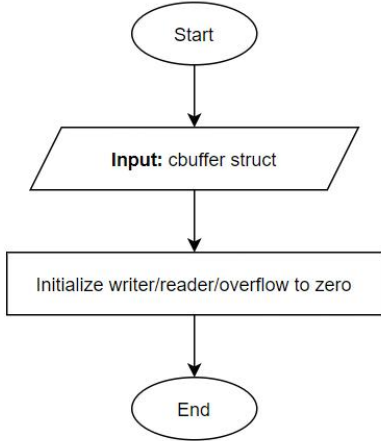
## 5. API REFERENCES

### 5.1. cb\_init

<b>API</b>	void <b>cb_init</b> (cbuffer_t *cb, void *buf, uint32_t size)
<b>Purpose</b>	Initialize a new Circular Buffer
<b>Description</b>	<p><b>cb_init</b> 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).</p> <p><b>Note:</b> All Cbuffer must be initialized before any kind of action involved that certain buffer.</p>
<b>Flowchart</b>	 <pre> graph TD     Start([Start]) --&gt; Input[/Input: cbuffer struct, buf, size/]     Input --&gt; Assign[Assign the address of circular buffer on RAM (cb-&gt;data = buf)]     Assign --&gt; Init[Initialize writer/reader/overflow to zero]     Init --&gt; Active[Mark cbuffer active status as true]     Active --&gt; End([End]) </pre> <p>The flowchart illustrates the steps for initializing a circular buffer. It begins with a 'Start' terminal, followed by an input step receiving 'cbuffer struct, buf, size'. The process then assigns the address of the circular buffer on RAM (cb-&gt;data = buf), initializes the writer/reader/overflow to zero, marks the cbuffer active status as true, and finally reaches the 'End' terminal.</p> <p>Figure 4. Flowchart of cb_init API</p>

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

## 5.2. cb\_clear

<b>API</b>	void <b>cb_clear</b> (cbuffer_t *cb)
<b>Purpose</b>	Clear counter and data of assigned Cbuffer.
<b>Description</b>	Reset Cbuffer's reader, writer, and overflow counter, technically, ignore and allow overwrite of all currently valid data in Cbuffer.
<b>Flowchart</b>	 <pre> graph TD     Start([Start]) --&gt; Input[/Input: cbuffer struct/]     Input --&gt; Process[Initialize writer/reader/overflow to zero]     Process --&gt; End([End]) </pre> <p>The flowchart illustrates the process of clearing a circular buffer. It begins with a 'Start' terminal, followed by an input step 'Input: cbuffer struct'. The main process is 'Initialize writer/reader/overflow to zero', which leads to an 'End' terminal.</p> <p>Figure 5. Flowchart of cb_clear API</p>



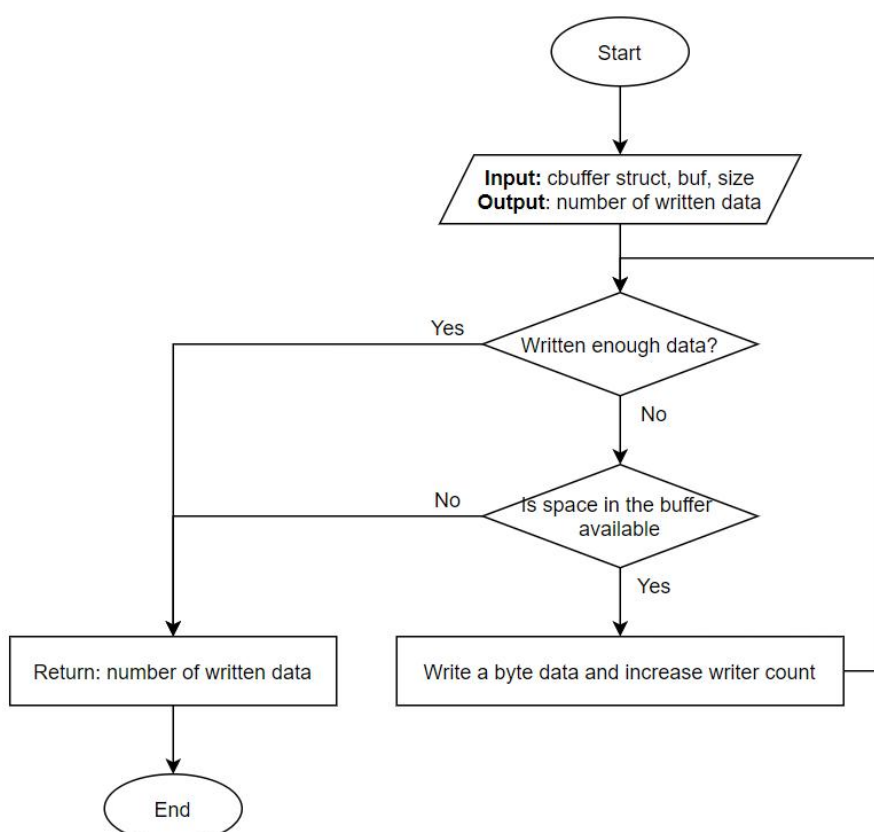
WORK FORM		
Document Title	Document Description	Version No.
	Software Detailed Design, Circular Buffer on RAM	0

### 5.3. cb\_read

<b>API</b>	uint32_t <b>cb_read</b> (cbuffer_t *cb, void *buf, uint32_t nbyte)
<b>Purpose</b>	Read a number of byte from Cbuffer and raise the reader count.
<b>Description</b>	Read “ <i>n_byte</i> ” number of data from Cbuffer and store it in “ <i>buf</i> ” (an external buffer ). The actual read bytes will be returned.
<b>Flowchart</b>	<pre> graph TD     Start([Start]) --&gt; IO[/Input: cbuffer struct, size Output: buf, number of read data/]     IO --&gt; D1{Received enough data?}     D1 -- Yes --&gt; R1[Return: number of read data]     R1 --&gt; End([End])     D1 -- No --&gt; D2{Is any data available}     D2 -- No --&gt; R1     D2 -- Yes --&gt; R2[Read a byte data and increase reader count]     R2 --&gt; D3{Is address of buf valid}     D3 -- No --&gt; D1     D3 -- Yes --&gt; R3[Copy a byte data to buf]     R3 --&gt; D1 </pre> <p>Figure 6. Flowchart of cb_read API</p>

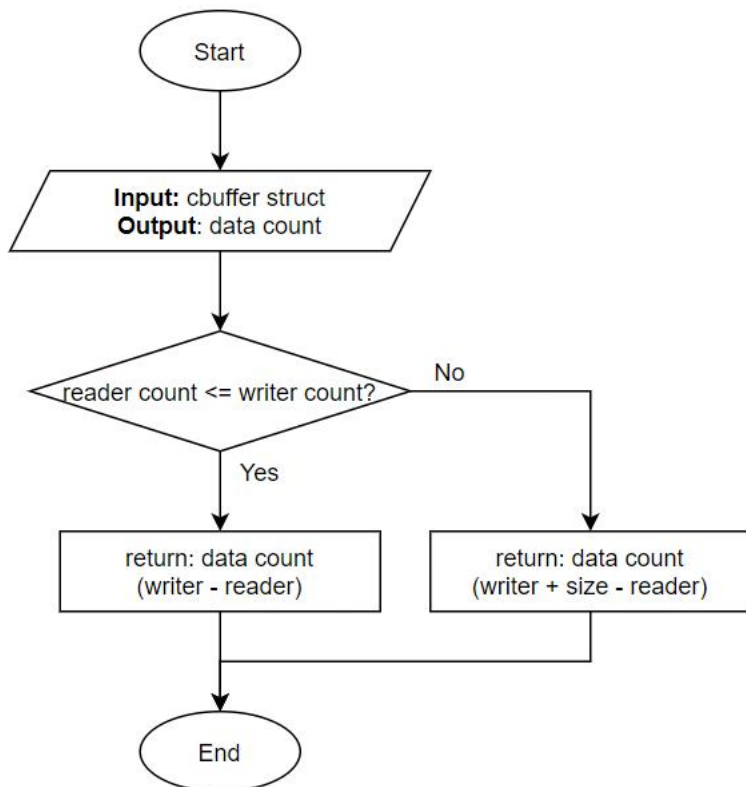
WORK FORM		
Document Title	Document Description	Version No.
	Software Detailed Design, Circular Buffer on RAM	0

#### 5.4. cb\_write

<b>API</b>	uint32_t <b>cb_write</b> (cbuffer_t *cb, void *buf, uint32_t nbyte)
<b>Purpose</b>	Write a number of byte to Cbuffer and raise the writer count.
<b>Description</b>	Write " <i>n_byte</i> " number of data from " <i>buf</i> " and store it in cbuffer. The actual written bytes will be returned.
<b>Flowchart</b>	 <pre> graph TD     Start([Start]) --&gt; Input[/Input: cbuffer struct, buf, size Output: number of written data/]     Input --&gt; Decision1{Written enough data?}     Decision1 -- Yes --&gt; Return[Return: number of written data]     Decision1 -- No --&gt; Decision2{Is space in the buffer available?}     Decision2 -- No --&gt; Return     Decision2 -- Yes --&gt; Write[Write a byte data and increase writer count]     Write --&gt; Decision1     Return --&gt; End([End]) </pre> <p>The flowchart illustrates the logic of the <code>cb_write</code> API. It begins with a 'Start' terminal, followed by an input/output block specifying the inputs as 'cbuffer struct, buf, size' and the output as 'number of written data'. The process enters a loop starting with the decision 'Written enough data?'. If 'Yes', it proceeds to 'Return: number of written data' and then to the 'End' terminal. If 'No', it moves to the decision 'Is space in the buffer available?'. If 'No' here, it also proceeds to 'Return: number of written data' and 'End'. If 'Yes', it executes 'Write a byte data and increase writer count', which then loops back to the 'Written enough data?' decision point.</p> <p>Figure 7. Flowchart of cb_write API</p>

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

### 5.5. cb\_data\_count

<b>API</b>	uint32_t <b>cb_data_count</b> (cbuffer_t *cb)
<b>Purpose</b>	Calculate readable bytes from cbuffer.
<b>Description</b>	<b>cb_data_count</b> calculate the number of readable bytes base on reader and writer count.
<b>Flowchart</b>	 <pre> graph TD     Start([Start]) --&gt; Input[/Input: cbuffer struct Output: data count/]     Input --&gt; Decision{reader count &lt;= writer count?}     Decision -- Yes --&gt; Process1[return: data count (writer - reader)]     Decision -- No --&gt; Process2[return: data count (writer + size - reader)]     Process1 --&gt; End([End])     Process2 --&gt; End </pre> <p>The flowchart illustrates the logic for calculating readable bytes in a circular buffer. It begins with a 'Start' terminal, followed by an input/output block specifying 'Input: cbuffer struct' and 'Output: data count'. A decision diamond checks if 'reader count &lt;= writer count?'. If 'Yes', the process block calculates 'return: data count (writer - reader)'. If 'No', the process block calculates 'return: data count (writer + size - reader)'. Both paths converge to an 'End' terminal.</p> <p>Figure 8. Flowchart of cb_data_count API</p>

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

5.6.   cb\_space\_count

API	uint32_t <b>cb_space_count</b> (cbuffer_t *cb)
Purpose	Calculate the number of data spaces can be written to cbuffer.
Description	<b>cb_space_count</b> uses the size of cbuffer minus the number of readable bytes to determine the remaining space.
Flowchart	<div> <pre> graph TD     Start([Start]) --&gt; IO[/Input: cbuffer struct Output: space count/]     IO --&gt; Process[return: space count (size - data count - 1)]     Process --&gt; End([End]) </pre> </div> <p>Figure 9. Flowchart of cb_space_count API</p>