CIRCULAR BUFFER IMPLEMENTATION USING ECLIPSE TOOL

Padmashree S¹, Chandana H C², Krupa V S³, L Kanvitha⁴, Mahitha H K⁵

Associate Professor, Department of Electronics and Communication Engineering, GSSSIETW, Mysuru, India¹
U.G. Student, Department of Electronics and Communication Engineering, GSSSIETW, Mysuru, India²
U.G. Student, Department of Electronics and Communication Engineering, GSSSIETW, Mysuru, India³
U.G. Student, Department of Electronics and Communication Engineering, GSSSIETW, Mysuru, India⁴
U.G. Student, Department of Electronics and Communication Engineering, GSSSIETW, Mysuru, India⁵

ABSTRACT:

We introduce a simple and efficient method for implementing the circular buffer using Eclipse Juno tool in C++ programming language. The initialization of buffer and updating is done dynamically during run time. Operations such as insertion, deletion, resizing are also implemented.

Keywords: Dynamic memory allocation, fixed length, New and Delete, FIFO, Dynamic resize, Queue, Eclipse

[1] INTRODUCTION

In general the term circular buffer refers to an area in memory which is used to store incoming data. When buffer is filled, new data is written starting at the beginning of the buffer and overwrite in the hold¹⁹.
The circular buffer is an efficient mechanism for maintaining a moving list of elements in an ordered fashion. The circular buffer is especially designed to provide fixed capacity storage, when its capacity is exhausted, newly inserted elements will cause elements to be overwritten at the beginning. The useful property of a circular buffer is that it does not need to have its elements shuffled around when one is consumed. In other words, the circular buffer is well-suited as a FIFO buffer.

Circular queue is a linear data structure. In circular queue the last node is connected back to the first node to make a circle. Circular buffering makes a good implementation strategy for a queue that has fixed maximum size. However, expanding a circular buffer requires shifting memory. Some implementations of the circular buffer use fixed-length elements that are bigger than 8-bit bytes—16-bit integers for audio buffers, 53-byte ATM cells for telecom buffers, etc. Each item is contiguous and has the correct data alignment, so software reading and writing these values can be faster than software that handles non-contiguous and non-aligned values.

Fixed-sized compressed circular buffers use an alternative indexing strategy based on elementary number theory to maintain a fixed-sized compressed representation of the entire data sequence. Most embedded programmers come to realize that majority of the code that they write in a day are either related to state machines or circular buffers or queues. In this article we will have a look at circular buffers and how you can implement them in low memory devices using Eclipse Juno tool. Eclipse Juno is an open source started as a proprietary IBM product which is best known as an Integrated Development Environment (IDE). Eclipse Juno is originally designed for Java, now supports many other languages.

[2] RELATED WORK

The highlighting feature of circular buffer is FIFO mechanism. Some of the methods used were as follows.

A. Circular buffers implementation using embedded C

Most embedded programmers come to realize that majority of the code that they write in a day are either related to state machines or circular buffers or queues. Circular buffer is a kind of array that will loop back to 0 after it reaches the maximum number of bytes in the buffer. This is done by having two pointers to the array one points to the “head” and other points to the “tail” of the buffer. As the data is added to the buffer, the head pointer moves up and as the data is being removed (read) the tail pointer moves up. This is implementation dependent and varies with perspective.

The next big thing about circular buffers is that there is no “clean way” to differentiate the buffer full and buffer empty cases. This is because at both cases, head is equal to tail. There is lot of ways/workarounds to deal with this issue. In this method, there are two critical cases that have to be considered while implementing a circular buffer, Head is equal to tail --the buffer is empty (Head+1) is equal to tail—the buffer is full. The essence is that every time you try to push, you check for “is buffer- full” condition and every time there is pop, you check for “is-buffer-empty”.
B. Circular buffer implementation using VHDL

In this module (in both Verilog and VHDL) they have used a First-in-First-out (FIFO) buffer module commonly used to buffer variable-rate data transfers or to hold/buffer data used in digital communication and signal processing algorithms. For example, a FIFO module can be used as circular buffer or delay line in a FIR filter.

If available, the tools will use the embedded block RAM resources within the FPGA[7]. The FIFO.vhd and FIFO_v.v modules are verified in test benches by writing and reading values to and from the FIFO while observing the RAM data and the condition of the output flags[8].

C. Circular buffer implementation using assembly level language.

The high speed and ultra-high speed microcontroller families have many features that greatly simplify the implementation of software FIFO. All of these features seek to minimize the software overhead associated with storing and retrieving the data. First among the feature is the extremely high-speed nature Dallas semiconductor Maxim microcontrollers. Their enhanced 4 clock and 1 clock per-machine cycle cores minimize the time spent servicing the buffer, and by extension allow the main application to run faster.

Another significant feature is the dual data pointers. The original 8051 only had only one data pointer which made it difficult to simultaneously implement input and output pointers. Each time the system needed to switch between pointers it would have to save off the current value and load the value of the other pointer. This introduces a significant delay when attempting to quickly move the data in and out of the buffer. With two data pointers one can be assigned to the input (insert) pointer and other to the output (remove) pointer of the buffer, eliminating the delay associated with juggling the pointers. In addition, some microcontrollers incorporate enhanced data pointers that can automatically increment the data pointer following the execution of certain data pointer-related instruction.

[3] PROPOSED METHOD

The circular buffer is an efficient mechanism for maintaining a moving list of elements in an ordered fashion. The circular buffer is especially designed to provide fixed capacity storage, when its capacity is exhausted, newly inserted elements will cause elements to be overwritten at the beginning. The useful property of a circular buffer is that it does not need to have its elements shuffled around when one is consumed. In other words, the circular buffer is well-suited as a FIFO buffer and the Eclipse Juno tool is used for implementation of circular buffer. Eclipse in an open-source Integrated Development Environment (IDE) supported by IBM. Eclipse is popular for Java project development. It also supports C/C++, PHP, Python, Perl and other web project developments via extensible plug-ins. Eclipse is cross-platform and runs under Windows, Linux and Mac OS. The CDT Project provides a fully functional C and C++ IDE based on Eclipse platform[9].

A circular buffer first starts empty and of some predefined length. Elements are inserted into the queue. If buffer is full the initially entered element is moved out and the recently entered element occupies the last position. The element can also be deleted from the specified location and the successive elements are shifted[10]. The additional feature is that the size of
the buffer can be dynamically resized so that the recent elements are retained if the size of the buffer is decreased\cite{11}.

The methodology of circular buffer is as follows, 
The figure2 shows the basic flow diagram for the proposed work. 
It is elaborated as follows.

1. **Overwrite Operation**
   Overwrite operation occurs when an element is inserted into a full circular buffer - the old element is being overwritten by the new one\cite{12}. It may be either a destruction of the original element and a consequent in place construction of a new element or it may be an assignment of a new element into an old one\cite{13}. The circular buffer implements assignment because it is more effective. . If there is a requirement for elements to be destructed/ constructed instead of being assigned insert operation is used\cite{14}.

2. **Writing to a Full Buffer**
   There are several options how to cope with the case if a data source produces more data than can fit in the fixed-sized buffer:
   - If the latest data is the most important, write over the oldest data\cite{15}.
   - Let the producer to be responsible for checking the size of the buffer prior writing into it.
   The circular buffer was designed and optimized to be circular (which means overwriting the oldest data when full).

3. **Reading/Removing from Buffer**
   The read operation enters the elements in to the buffer; if buffer size is more than fixed length then elements entered will be shifted in FIFO manner\cite{16}. Particular element can be removed from buffer if it is not required. Since dynamic memory allocation is used, memory is allocated to elements during execution\cite{17}.
[4] EXPERIMENTAL RESULTS

According to the various review papers the implementation of circular buffer using Eclipse Juno was found to give the best results compared to the different methods.

This paper is expected to have the following results.

- Circular buffer is implemented with element insertion and deletion, resizing features using Eclipse tool in C++ programming language.
- A specialized feature of dynamic memory allocation and de-allocation is included.

[5] ACKNOWLEDGEMENT

First and foremost, we would like to express our sincere gratitude towards our guide Mrs. Padmashree S for providing excellent guidance and encouragement and giving us an opportunity to complete internship in Skanray technologies Healthcare Pvt ltd. We would like to express our sincere gratitude towards our Head of the Department of Electronics and Communication Engineering Prof. Dr. M V Sathyanarayana for his guidance and inspiration.
We would also like to thank our Principal Dr. Sumithradevi K A for providing all the facilities and a proper environment to work on the project.

[6] CONCLUSION

In this paper we have implemented circular buffer with the features of key insertion, deletion and resizing of buffer using eclipse tool in C++ programming language. This benefits every programmer to keep in track of all the data accessed by the user. This implementation of buffer includes a specialised feature of dynamic memory allocation and deallocation.

REFERENCES

[1] Method of management of a circular queue for asynchronous access, David J. Garcia, David P. Sonnier
[2] Data handling system with circular queue formed in memory, Pascal Regache
[5] Using eclipse as a tool-integration platform for software development, Zhihui Yang, Micheal Jiang, IEEE software
[8] A simple, fast and scalable non-blocking concurrent FIFO queue for shared memory multiprocessor systems, Philippas Tsigas
[10] System for accessing a space appended to a circular queue after traversing an end of the queue and upon completion copying data back to the queue, William R. Etcheverry.
[12] DSP based FIR filter design and circular buffer implementation, Hue Wen, Shu Ming Li.
[13] Computer system having a data buffering system which includes a main ring buffer comprised of a plurality of sub-ring buffers connected in a ring, Osamu Ichikawa.
[18] System for accessing a space appended to a circular queue after traversing an end of the queue and upon completion copying data back to the queue, William R. Etcheverry.
[19] Memory-based circular queue with local descriptive information to implement a storage area for filtering mpeg-2 packets that are distributed and/or processed under the control of a host microprocessor by a direct access mechanism, Alek Movshovich, Robert H. Hoem, Niranjan A. Puttaswamy, Brian Lai.
[20] Circular Buffered Switch Design with Wormhole Routing and Virtual Channels, Nan Ni, Marius Pirvu and Laxmi Bhuyan, Department of Computer Science, Texas A&M University.