**ECEN 5813 Principles of Embedded Software**

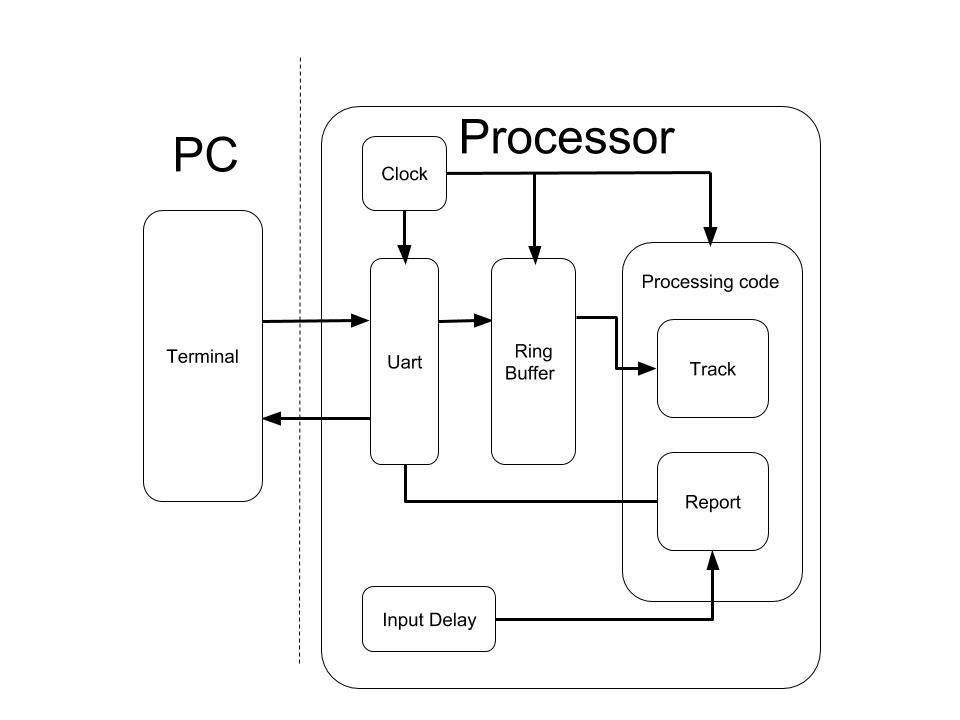
**Project 2 S2019 final report**

**Circular buffer, UART and interrupts**

Linfeng Li

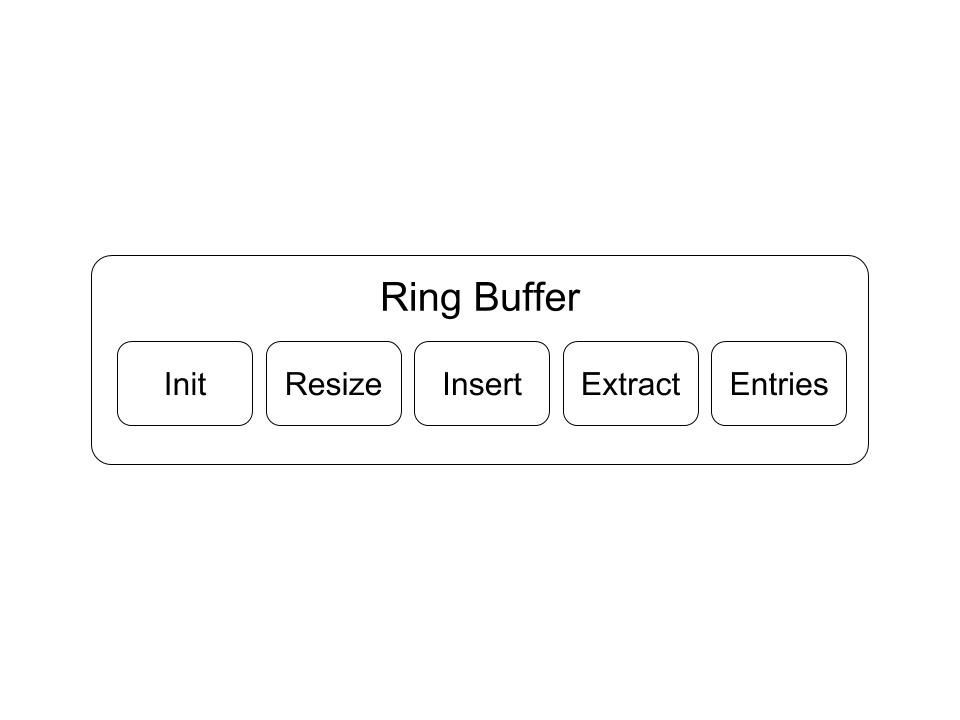
Wei Ting Chien

4/9/2019

**Part 1 : Block diagram and architecture**

The software diagram can be divided into two parts, PC and Processor. User will input the data stream with terminal software on PC side, and the terminal software will transfer those data to the processor by Uart in real time. While processor receive data, it will run “Ring Buffer” code to save data. And then those data in ring buffer will go through the processing code to generate a report. Moreover, a “Track” function is used in processing code in order to track ring buffer’s condition. To avoid any disturbance in report, “Input delay” function defines the delay time interval and the algorithm. Furthermore, a clock source is needed to synchronize all the peripherals and functions in processor.

|  |  |
| --- | --- |
| Components | Functionalities |
| Terminal | Communicate with Processor |
| Uart | Communicate with terminal and send data to ring buffer |
| Ring buffer | Manage the input and output data |
| Processing code | Monitor ring buffer and generate report |
| Input delay | Prevent report generation from any disturbance |
| Clock | Distribute the clock to core and peripherals |

 In ring buffer, there are 5 components, Init, Resize, Insert, Extract, and Entries.

|  |  |
| --- | --- |
| Components | Functionalities |
| Init | Initialization of the ring buffer with certain length. |
| Resize | Resize an exisiting buffer with certain length.  Data existed in the buffer would be maintained.  Only latest data would be maintained if new size cannot hold all existing data. |
| Insert | Insert one element into the buffer.  Buffer Full situation detection . |
| Extract | Extract one element from the buffer.  Buffer empty situation detection. |
| Entries | return the number of elements in the buffer that have been wrote but yet to read. |

## Questions

**Part2 : Circular buffers**

1. **Is your implementation thread safe? Why or why not?**

No. We use some global variables in the ring buffer, and those variables can be used by several threads. If any thread is interrupted by other thread, and both threads use the same variable, some errors will occur.

1. **What potential issues exist if the same buffer is used by both interrupt and non-interrupt code? How can these issues be addressed?**

If a buffer is used by non-interrupt and interrupt code, some error may occur when the interrupt is triggered, in the meanwhile, a buffer is being used. This non-interrupt operation will be disturbed, and the buffer may be messed up, as known as data racing.

To avoid this error, a interrupt blocking should be set in certain part of code where the sensitive variables are.

1. **How could you test these issues?**

**Part4 : UART device driver**

1. **For each implementation, what is the CPU doing when there are no characters waiting to be echoed? What is the behavior of the GPIO toggle in the non-blocking implementation?**

CPU will stop and wait for the user input in the blocking implementation. On the other hand, in non-blocking implementation, CPU can go through another procedure, i.e. remain data inside ring buffer without being read.

The GPIO toggle will keep going in non-blocking implementation.

1. **For each implementation trace the sequence of events that occur by listing, in order, the functions called from the point that a character sent to the FRDM board has been received until the point where the echoed character has been sent.**
2. **Comment on the interface for sending and receiving characters presented to the main() application code for blocking vs. non-blocking variation. Which variation is easier to code to?**

Blocking is easier to code to because we do not need to consider the priorities of different interrupts or functions. However, the blocking application code will make the program stop in one place until the function complete. On the other hand, the non-blocking implementation can let the processor do other tasks and wait for the specific function by polling flags. In this case, the order of polling should be considered.

**Part5 : Application**

1. **What is the CPU doing after the last character has been received and while the report is being printed?**

To cope with the onslaught, the application code has an input delay. When the onslaught stops, the state machine would wait for 50 ms and then generate a new report, which means putting the new report into the transmit ring buffer. This wait is a non-blocking one. The state machine would check whether the tick is over a pre-determined value. If it does, the state machine would switch to a new state and starting generate the new report. If it doesn’t, the state machine would return and run the pseudo-random number generator once.

While the report is being printed, there are several situations. If an onslaught happens, the received characters are stored into the receive ring buffer and read by the application state machine. If just several characters received, the state machine should be waiting for the 50 ms input delay. Since the ring buffer insertion cannot be interrupted by the extraction, The state machine would always check whether the previous report has already been completely transmitted by checking whether the UART transmit interrupt is enabled. If UART transmitting interrupt is disabled, which means that the ISR has already read all of the data of the previous report, sent via UART, disabled the UART transmit interrupt. Thus, a new report could be generated and stored in the transmit ring buffer.

1. **Baud rate aside, what limits the rate at which the application can process incoming characters? What happens when characters come in more quickly than they can be processed?**

1. **How does the size of the circular buffer affect report output behavior (especially during an onslaught)? What is an appropriate buffer size to use for this application? Why?**

## Appendix