Buffering

ENCE361 Embedded Systems 1

Course Coordinator: Ciaran Moore (ciaran.moore@Canterbury.ac.nz)

Lecturer: Le Yang (le.yang@canterbury.ac.nz)

Department of Electrical and Computer Engineering

Where we're going today

- **Introduction to buffering**
- Circular buffer and example code
- Double buffer and example code
- Homework

Introduction to Buffering (1)

- Buffer refers to a region of physical memory for storing data *temporarily*
- When do we need a buffer?
  - Data consumption is not synchronous with data producing
    - In temperature monitoring, regular sensor readings is buffered before transmission/processing
    - Online video playback …
  - Processing data in batches
    - Signal averaging buffers most recent *M* samples, so does finite impulse response (FIR) filtering

$$z(nT_S) = \frac{1}{M}\sum_{m=0}^{M-1} y((n-m)T_S)$$

    - N-point Discrete Fourier transform (DFT) needs to buffer *N* samples

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi kn}{N}}, k = 0,1,2,\dots,N-1$$

Introduction to Buffering (2)

- Some key design considerations for buffering
  - Data type
  - Buffer size

  - Write a data sample/a few data samples to the buffer
  - Read a data sample/a few data samples from the buffer

  - Memory allocation
    - Dynamic allocation vs. static allocation

Where we’re going today

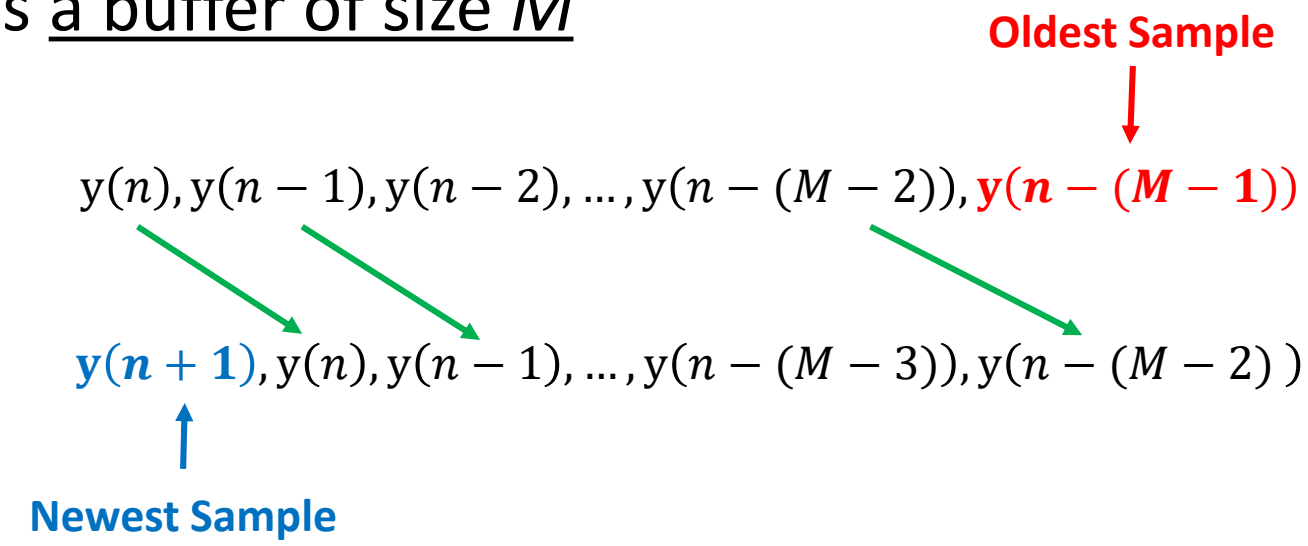
- Introduction to buffering
- **Circular buffer and example code**
- Double buffer and example code
- Homework

Circular Buffer (1)

- Signal averaging revisited

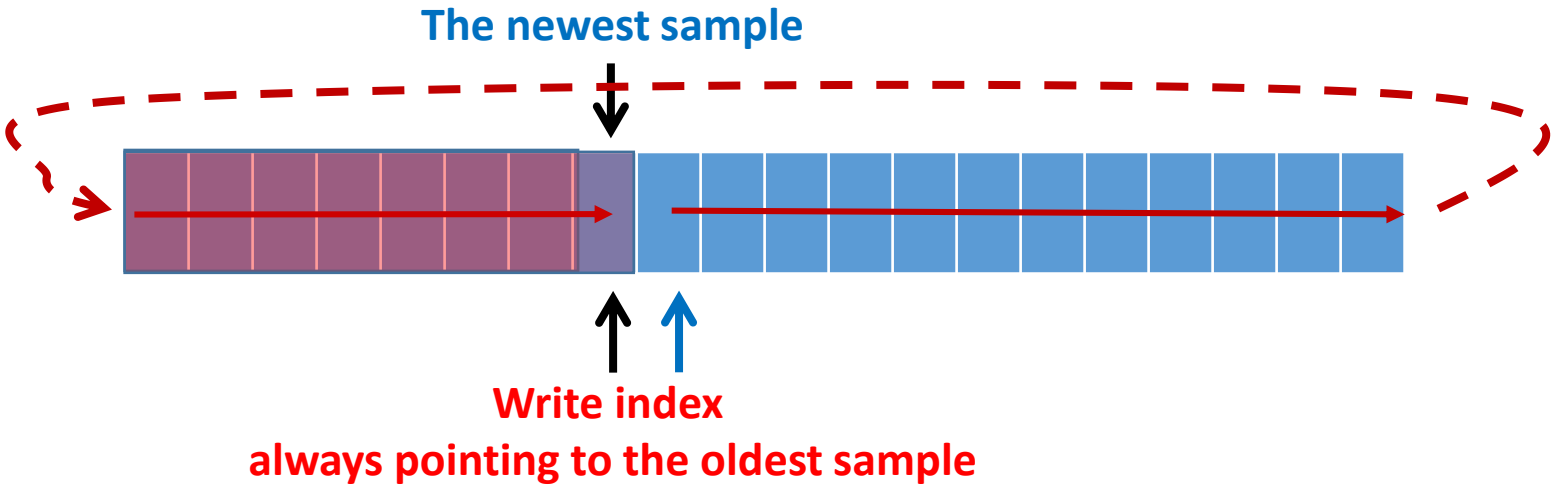
$$z(nT_S) = \frac{1}{M}\sum_{m=0}^{M-1} y((n-m)T_S)$$

- Signal averaging needs a buffer of size *M*
  - At time n, we use
  - At time n+1, we use



Circular Buffer (2)

- May use a shift register that
  - Shift the first M-1 elements towards the end of the buffer and write the new sample to the beginning of the buffer
- Circular buffer for data stream
  - Overwrite the oldest sample with the newest one
  - Maintain the correct temporal order of all the samplings without shifting them



Example code: circBufT.h in ADCdemo1.c (1)

```
#ifndef CIRCBUFT_H_
#define CIRCBUFT_H_
// ***********************************************************
// circBufT.h
//
// Support for a circular buffer of uint32_t values on the Tiva processor.
// Created by P.J. Bones, UC ECE, Updated by Le Yang
// ***********************************************************
#include <stdint.h>

// ***********************************************************
// Buffer structure
  typedef struct {
            uint32_t size;        // Number of entries in buffer
            uint32_t windex;      // Index for writing (0 – size-1)
            uint32_t rindex;      // Index for reading (0 – size-1)
            uint32_t *data;       // Pointer to the starting address of the buffer
  } circBuf_t;
```

Example code: circBufT.h in ADCdemo1.c (2)

```
// ******************************************************
// initCircBuf: Initialise the circBuf instance. Reset both indices to  the start of the buffer.
// Dynamically allocate and clear the memory and return a pointer for the data.
// Return NULL if  allocation fails.
uint32_t *initCircBuf (circBuf_t *buffer, uint32_t size);

// ******************************************************
// writeCircBuf: insert entry at the current windex location, advance windex, modulo (buffer size).
void writeCircBuf (circBuf_t *buffer, uint32_t entry);

// ******************************************************
// readCircBuf: return entry at the current rindex location, advance rindex, modulo (buffer size).
uint32_t readCircBuf (circBuf_t *buffer);

// ******************************************************
// freeCircBuf: Releases the memory allocated to the buffer data, sets pointer to NULL and other fields to 0. The
buffer can be re-initialised by another call to initCircBuf().
void freeCircBuf (circBuf_t *buffer);
```

Example code: circBufT.c in ADCdemo1.c

```
// ***********************************************************
// circBufT.c
// Support for a circular buffer of uint32_t values on the Tiva processor.
// Created by P.J. Bones, UC ECE, Updated by Le Yang, UC ECE
// ***********************************************************
#include <stdint.h>
#include "stdlib.h"
#include "circBufT.h"

// ***********************************************************
// writeCircBuf: insert entry at the current windex location, advance windex, modulo (buffer size).
void writeCircBuf (circBuf_t *buffer, uint32_t entry)
{
        buffer->data[buffer->windex] = entry;       // Overwrite the oldest sample with the newest one
        buffer->windex++;                           // Advance the windex to the 'right'
        if (buffer->windex >= buffer->size)
          buffer->windex = 0;                       // Reset the windex to the start of the buffer
}
```

Example code: ADCdemo1.c (1)

```
// ADCdemo1.c - Simple interrupt driven program which samples with AIN0
. . . .
#include "circBufT.h"

// Constants
//*****************************************************************************
#define BUF_SIZE 10                     // Buffer size
#define SAMPLE_RATE_HZ 10

// Global variables
//*****************************************************************************
static circBuf_t g_inBuffer;            // uint32_t buffer of size BUF_SIZE (intervals)
volatile static uint32_t g_u32IntCnt;   // Counter for interrupts
. . . .
int main(void)
{
……
        initCircBuf (&g_inBuffer, BUF_SIZE);
. . . .
```

Example code: ADCdemo1.c (2)

```
// ***************************************************************************
// The handler for the ADC conversion complete interrupt.
// Writes to the circular buffer.
//***************************************************************************
void  ADCIntHandler(void)
{
        uint32_t ulValue;

        // Get the single sample from ADC0.  ADC_BASE is defined in inc/hw_memmap.h
        ADCSequenceDataGet(ADC0_BASE, 3, &ulValue);
        //
        // Place it in the circular buffer (and advance the write index (windex))
        writeCircBuf (&g_inBuffer, ulValue);
        //
        // Clean up, clearing the interrupt
        ADCIntClear(ADC0_BASE, 3);
}
```

Example code: ADCdemo1.c (3)

- Circular buffer of length BUF_SIZE is used to calculate the average of most recent BUF_SIZE samples (signal averaging)

```
while (1)
  {
      // Background task: calculate the mean of the values in the
     // circular buffer and display it, together with the number of  obtained samples.
     sum = 0;
     for (i = 0; i < BUF_SIZE; i++)
         sum = sum + readCircBuf (&g_inBuffer);

     // Calculate and display the rounded mean of the buffer contents
     displayMeanVal ((2 * sum + BUF_SIZE) / 2 / BUF_SIZE, g_ulSampCnt);

     SysCtlDelay (SysCtlClockGet() / 6);  // Update display at ~ 2 Hz
  }
```

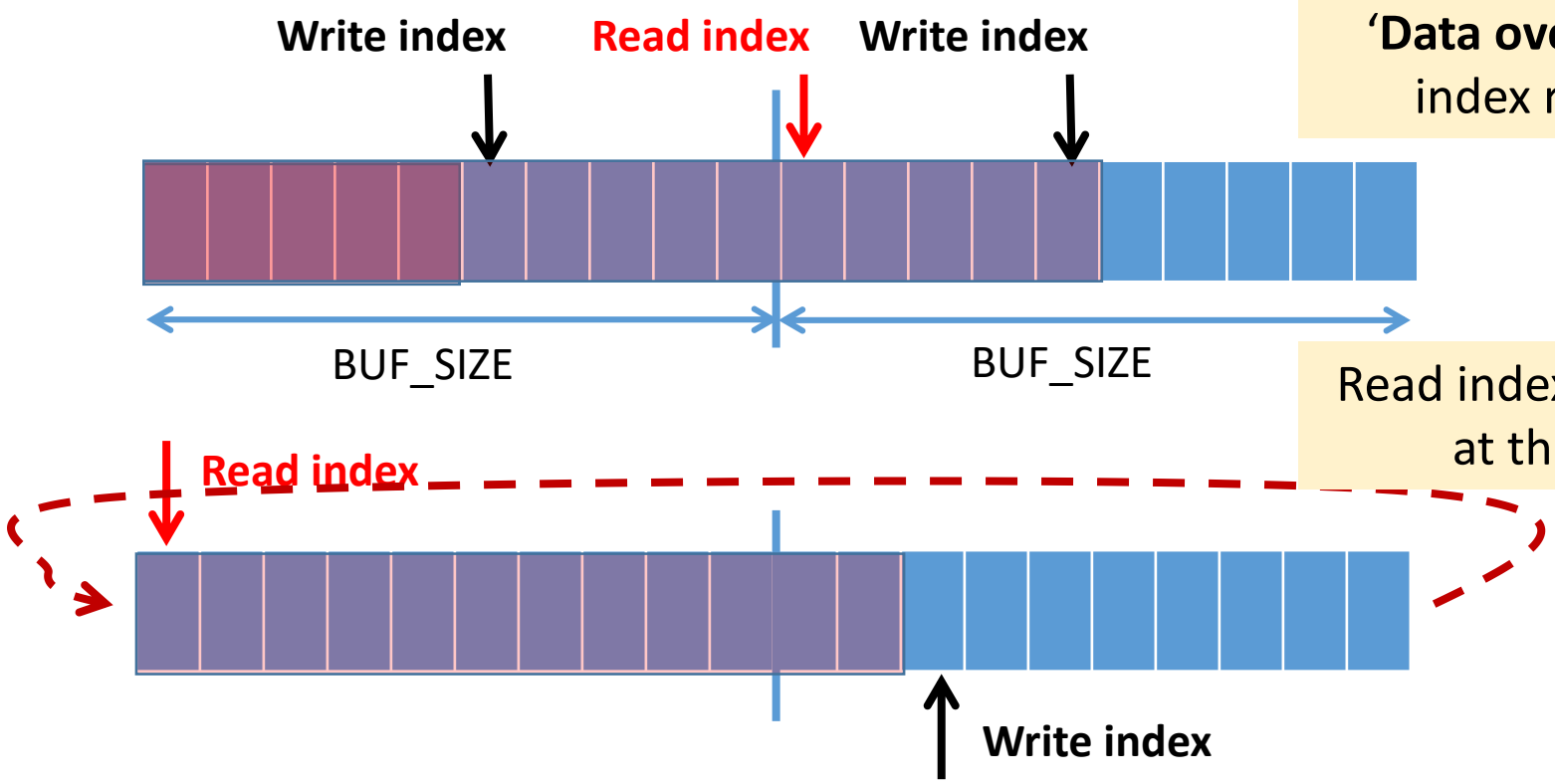
What happens if this calculation is interrupted?

Where we’re going today

- Introduction to buffering
- Circular buffer and example code
- **Double buffer and example code**
- Homework

Double Buffer

- Provide protection to buffered contents under interrupt conditions
- Cascade two buffers with same size to form a larger circular buffer
  - Read the data from one buffer in a block at a times while writing data to the other



'**Data overrun**' occurs if write index reaches read index

Read index is **always positioned** at the start of a block

Example Code for Double Buffer (1)

```
// ************************************************
// dbleBuf.h   Supports a double buffer of int32_t on Tiva

typedef struct {       // Buffer structure
  int32_t size;        // Number of entries in ½ buffer
  int32_t windex;      // index for writing, mod(2*size)
  int32_t rindex;      // index for reading, mod(2*size)
  int32_t *data;       // pointer to the starting address of the double buffer
} dbleBuf_t;

int32_t *initDbleBuf (dbleBuf_t *buffer, uint32_t size);

void writeDbleBuf (dbleBuf_t *buffer, int32_t entry);

int readDbleBuf (dbleBuf_t *buffer, int32_t *array);

void freeDbleBuf (dbleBuf_t *buffer);
```

Example Code for Double Buffer (2)

```
// ***************************************************
// readDbleBuf: return a complete sub buffer contents,  advance rindex.
// Return true (1) if data overrun is detected, otherwise false (0).

int readDbleBuf (dbleBuf_t *buffer, int32_t *array)
{
  int overrun = (buffer->windex >= buffer->rindex) &&
                !(buffer->windex >= buffer->rindex + buffer->size);     // Detect data overrun

   int i;

  for (i = 0; i < buffer->size; i++, (buffer->rindex)++)                // Read the data in one buffer into array
      array[i] = buffer->data[buffer->rindex];

  if (buffer->rindex >= 2*buffer->size)                                 // Circular buffering
    buffer->rindex = 0;
  return overrun;
}
```

Other Buffers

- Triple buffer?
  - Might be necessary if more protection to the buffered data under interrupt conditions is desired
  - Better protection at the cost of longer delay
- First-in, first out (FIFO) buffer
  - Important for e.g., serial communications
  - Could be realized as a shift register

Homework

1. Using **circBufT.c** as a model, write C code to implement the function **initDbleBuf ()** whose prototype is given in Slide 16.

2. Write C code to implement the function **writeDbleBuf ()** which has the prototype given on Slide 16.

3. Could you use the functions prototyped in **circBufT.h** and implemented in **circBufT.c** to implement a FIFO buffer (Slide 17)? If so, how? If not, why not?

4. In the call to **displayMeanVal ()** on Slide 13, the expression in red finds a rounded value for the mean using only integer arithmetic. Comment on the order of operations in the expression, does the order matter?