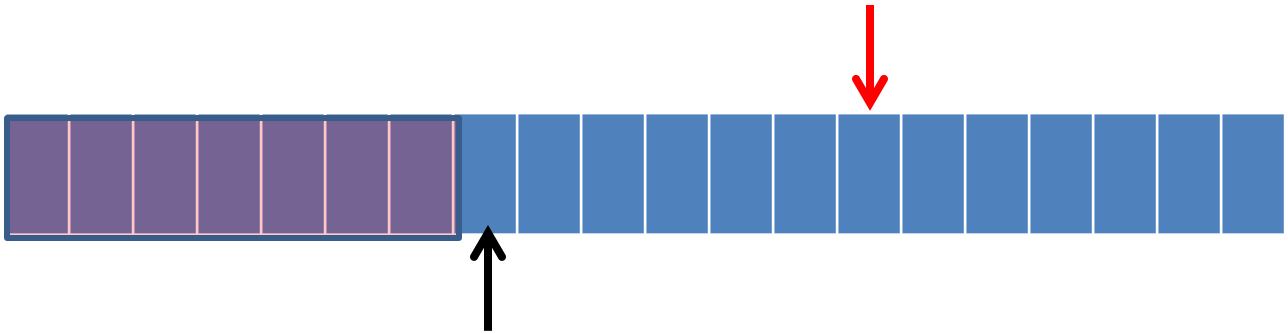
ENCE361 Embedded Systems 1

Buffering

Lecture 9 2018



Buffering

Example code relevant to this lecture and to Project Milestone 1:

ADCdemo1.c

circBufT.h

circBufT.c

ADCdemo1.c *Extract 1*

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

// Constants
//*****************************************************************
#define BUF_SIZE 10
#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)
{
        uint16_t i;      int32_t sum;

        initClock ();
        initADC ();
        initDisplay ();
        initCircBuf (&g_inBuffer, BUF_SIZE);
. . . .
```

ADCdemo1.c *Extract 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 (advancing write index)
        writeCircBuf (&g_inBuffer, ulValue);
        //
        // Clean up, clearing the interrupt
        ADCIntClear(ADC0_BASE, 3);
}
```

At each interrupt for conversion complete, a new sample is written to the circular buffer

Circular buffer

In **ADCdemo1.c**, the circular buffer of length *N* is used to make it straightforward to calculate an average of the most recent *N* samples.

Is that the best way?

Are there alternative methods?

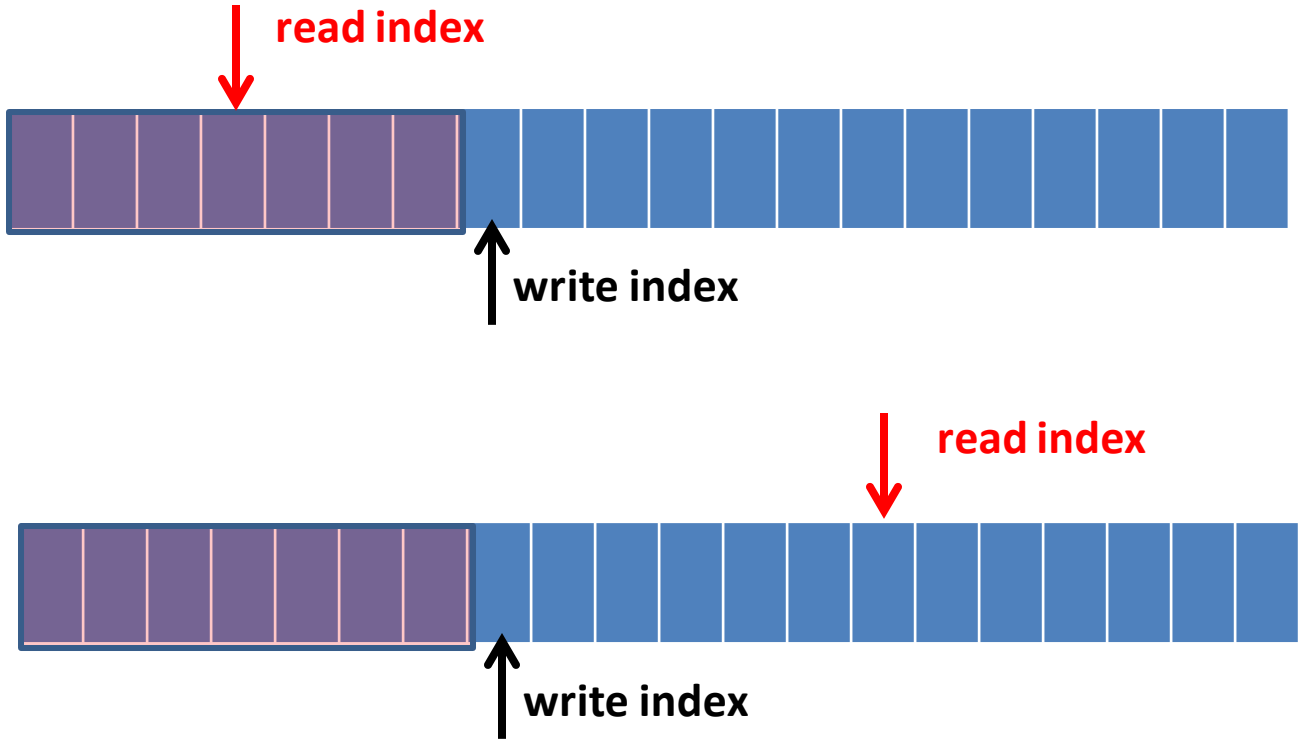
```
while (1)
{
    //
    // Background task: calculate approx. mean of the values in the
    // circular buffer and display it, together with the sample number.
    sum = 0;
    for (i = 0; i < BUF_SIZE; i++)
        sum = sum + readCircBuf (&g_inBuffer);
    // Calculate and display the rounded mean of the buffer contents
    displayMeanVal ((2 * sum + BUF_SIZE) / 2 / BUF_SIZE, g_ulSampCnt);

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

ADCdemo1.c *Extract 3*

What happens if an interrupt occurs while this calculation is performed?

Circular buffering



Producer – Consumer

Producing and consuming do not have to be synchronous

A circular buffer module - header

```
#ifndef CIRCBUFT_H_
#define CIRCBUFT_H_

// ***************************************************
//
// circBufT.h
//
// Support for a circular buffer of uint32_t values on the
//  Tiva processor.
// P.J. Bones UCECE
// Last modified:  7.3.2017
//
// ***************************************************
#include <stdint.h>

// ***************************************************
// Buffer structure
typedef struct {
        uint32_t size;    // Number of entries in buffer
        uint32_t windex;  // index for writing, mod(size)
        uint32_t rindex;  // index for reading, mod(size)
        uint32_t *data;   // pointer to the data
} circBuf_t;
```

```
// ***************************************************
// initCircBuf: Initialise the circBuf instance. Reset both indices to
// the start of the buffer.  Dynamically allocate and clear the 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). The function deos not check
// if reading has advanced ahead of writing.
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
// re initialised by another call to initCircBuf().
void
freeCircBuf (circBuf_t *buffer);
```

Header, cont.

A circular buffer module - implementation

```
// ***************************************************
// circBufT.c
// Support for a circular buffer of uint32_t values on the
//  Tiva processor.
// P.J. Bones UCECE
// Last modified:  8.3.2017
//
// ***************************************************
#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;
        buffer->windex++;
        if (buffer->windex >= buffer->size)
           buffer->windex = 0;
}
```

Buffering in general

Buffers are an important tool in handling data that becomes available as a result of *asynchronous* events, or which needs to be handled in batches, or when *consuming* is not synchronous with *producing*.

Consider the difference between operating a **finite impulse response (FIR) filter** on a sequence of sampled data and an **DFT (FFT) operation** on the same sequence.

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$

$$F(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi\frac{kn}{N}}$$

$x(n)$ is the stream of input samples

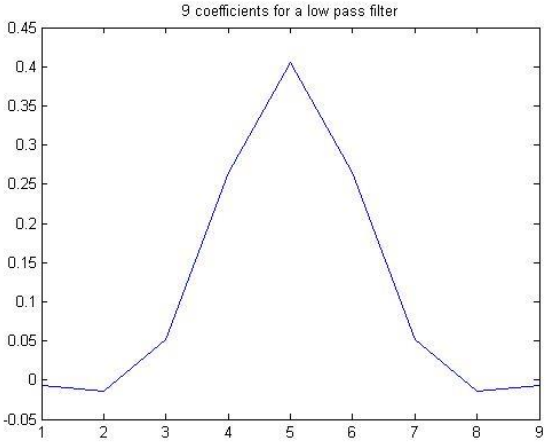
$y(n)$ is the stream of output samples

$h(m)$ is the set of filter coefficients

$F(k)$ is the discrete Fourier Transform for a set of N samples

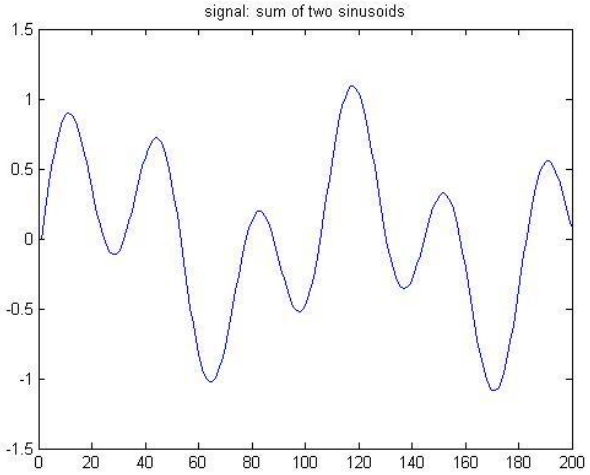
FIR filter

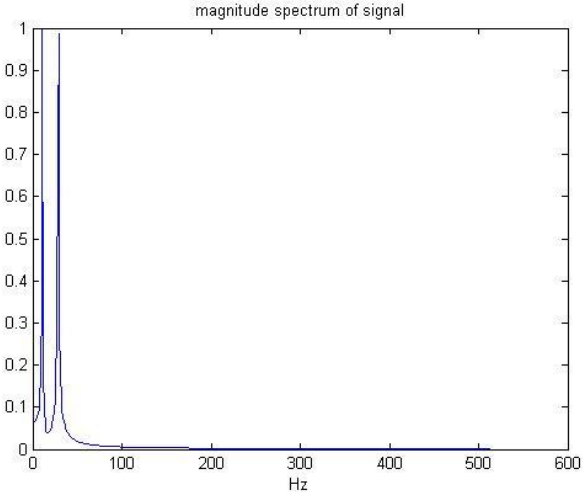
Low pass filter with 9 coefficients, cutoff frequency = 0.2 f_s



h = -0.0061 -0.0136 0.0512 0.2657 0.4057 0.2657 0.0512 -0.0136 -0.0061

DFT





Which type of buffer?

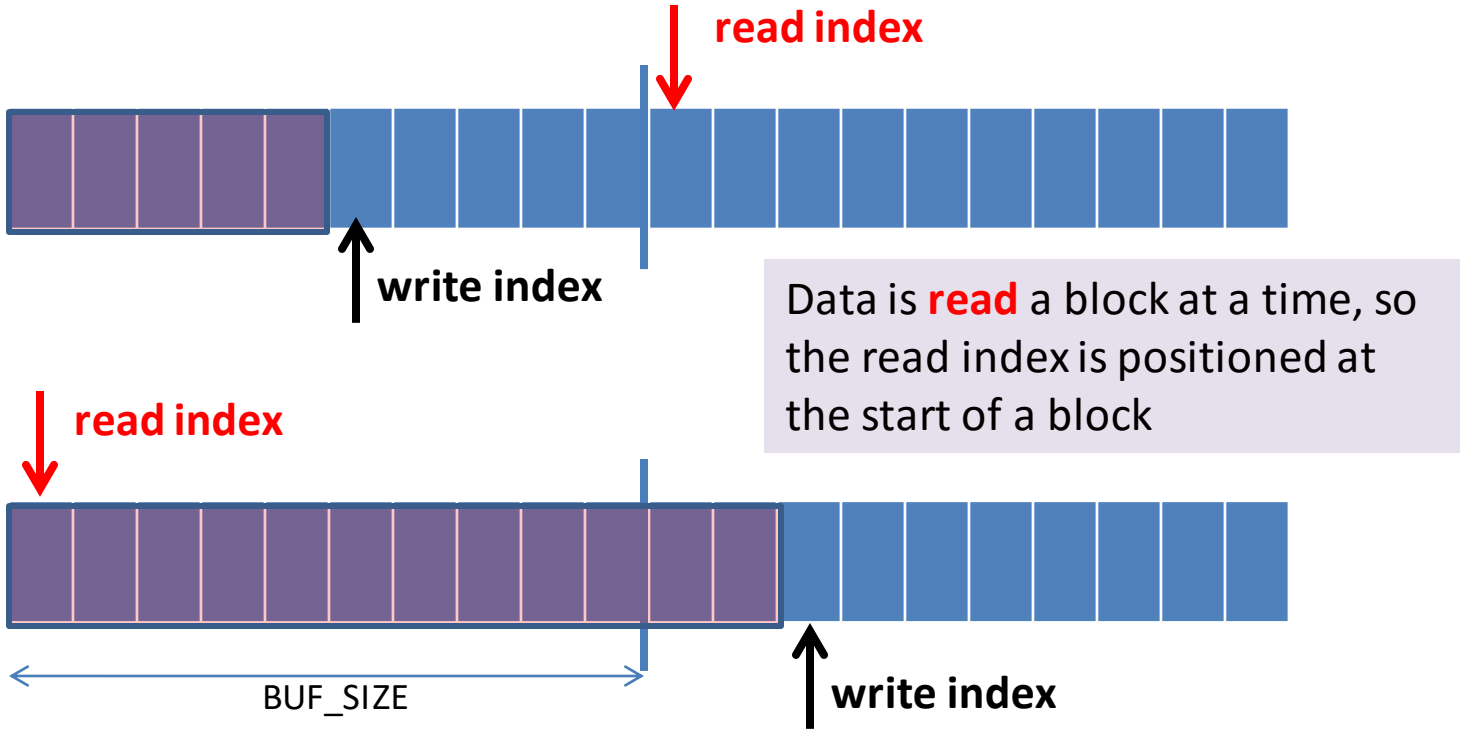
Buffers are an important tool in handling data that becomes available as a result of asynchronous events, or which needs to be handled in batches, or when consuming is not synchronous with producing.

Consider the difference between operating a **finite impulse response (FIR) filter** on a sequence of sampled data and an **DFT (FFT) operation** on the same sequence.

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$ - suits a circular buffer

$$F(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi\frac{kn}{N}}$$ - suits a double buffer

Double buffering



If the write index ever reaches the read index, 'data overrun' has occurred

read index = bufStart + bufNum * BUF_SIZE, where bufNum = {0,1}

Double buffering

```
// ********************************************************
// 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 data
} 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);
```

Double buffering

```
// ************************************************
// readDbleBuf: return a complete sub buffer contents,
// advance rindex. Return true (1) if 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);
    int i;

    for (i = 0; i < buffer->size; i++, (buffer->rindex)++)
        array[i] = buffer->data[buffer->rindex];
    if (buffer->rindex >= buffer->size)
        buffer->rindex = 0;
    return overrun;
}
```

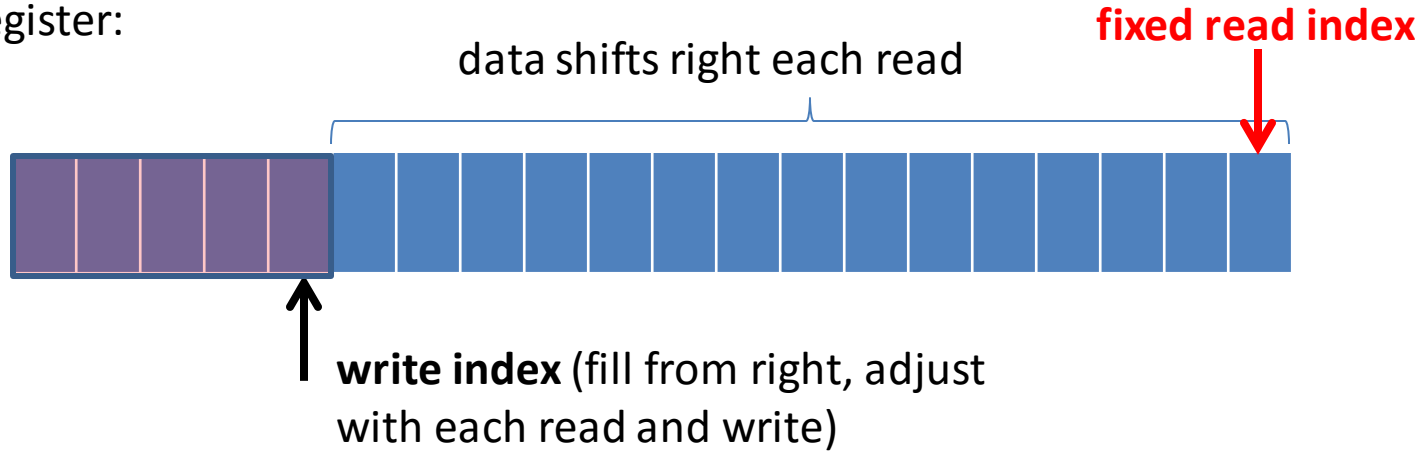
Other buffers

Triple buffer?

Could be required to give extra protection to the buffer contents under interrupt conditions.

FIFO buffer (often just called a "FIFO")

Important in serial communications. Can be implemented as a shift register:



Homework

1. Using **`circBufT.c`** as a model, write C code to implement the function **`initDbleBuf ()`** which has the prototype given on Slide 14.

2. Write C code to implement the function **`writeDbleBuf ()`** which has the prototype given on Slide 14.

3. Could you use the functions prototyped in **`circBufT.h`** and implemented in **`circBufT.c`** to implement a FIFO buffer (Slide 16)? If so, how? If not, why not?

4. In the call to **`displayMeanVal ()`** on Slide 5, an expression for the function argument achieves a rounded value for the mean using only integer arithmetic. Comment on the order of operations in the expression; does the order matter? Support your answer with examples.