NAME: _____ GRADE: _____

EE442 / EE592 Real-Time Digital Signal Processing
Quiz #4
Open Books, Notes, Calculators (no programs, no graphing)

Each question is worth 10 points unless otherwise noted.

Code listings of the various files used in programming the TI DSP are listed at the end of this quiz.

1. The cround() function convergently rounded values of type int for rehandling as type short. Explain precisely how cround() works—include information on the various hex values used and associated arithmetic; you may wish to illustrate your explanation with an example. Simply copying comments from the original code or stating the obvious ("accumulate a with 0x4000") is not sufficient.

cround.c
```
int cround(int a)
{
      if (a & 0x3fff)
            a += 0x4000;
      else
            a += (a>>1) & 0x4000;
      return a & 0xffff8000;
}
```

2. Why does every DSP routine (ccan, cdelay, cfir, etc...) have the line

```
#include "util.h"
```

but not also

```
#include "user_data.h"
```

Provide the value, z for the following codes.

3.
```
short x, y, z;

void main (void)
{
      x = 16000;
      y = 17000;
      z = x + y;
}
```

z = ?

4.
```
short x, y;
int z;

void main (void)
{
      x = 16000;
      y = 17000;
      z = x + y;
}
```

z = ?

5.
```
short x, y, z;

void main (void)
{
      x = 16384;
      y = 16384;
      z = (x * y)>>15;
}
```

z = ?

Write a C code for the following *real-time* programming tasks (Questions 6 – 8), assuming the TI dsk_app passcode. Identify by file name (user_data.h, initialize_program.c, process_signal.c) and line number where you would place your instructions. Printouts of the files are listed at the end of this quiz. The first programming task is done as an example.

6. Write a code which multiplies the right channel by $g = 0.7$.

Solution:

In user_data.h insert at line 12 the following line:

```
#define G 22938 /* short equiv for 0.7 */
```

In process_signal.c replace line 12 with the following line:

```
   *outputRight = cround(G * inputRight)>>15;
```

7. Write code (use cfir) which filters the right channel with an FIR filter whose coefficients are

$$h[n] = \begin{cases} 0.2, & 0 \le n \le 4 \\ 0, & \text{otherwise} \end{cases}.$$

Solution:

8. Write code (use ccan) which filters the right channel with an IIR filter whose coefficients are

$$b = \begin{bmatrix} 9598 & 19195 & 9598 \end{bmatrix}^T$$
$$a = \begin{bmatrix} 32767 & 0 & 5622 \end{bmatrix}^T .$$

Solution:

This page for EE592 students only.

9. Write a C code (non-real-time) which determines the odd parity bit for an ASCII code stored as 'letter' and writes the bit to the LSB of 'parity_bit', i.e.

```
void main (void)
{
    char letter = 'A';
    short parity_bit;

<YOUR CODE HERE>

    return 0;
}
```

Code Listings

```
1  /*
2  * USER_DATA.H
3  */
4
5  #ifndef USER_DATA
6  #define USER_DATA
7
8  void initialize_program();
9  void process_signal(short inputRight, short inputLeft, short *outputRight, short
       *outputLeft);
10
11 /* Global variables here as extern, initializations in initialize_program.c */
12
13 #include "util.h"
14
15 #endif
```

```
1  /*
2  * INITIALIZE_PROGRAM.C
3  */
4
5  #include "user_data.h"
6
7  /*************************************/
8  /* Global variable initializations here */
9  /*************************************/
10
11 void initialize_program()
12 {
13
14 }
```

```
1  /*
2  * PROCESS_SIGNAL.C
3  */
4
5  #include "user_data.h"
6
7  void process_signal(short inputRight, short inputLeft, short *outputRight, short
       *outputLeft)
8  {
9  /*******************************/
10 /* Process right channel sample */
11 /*******************************/
12 *outputRight = inputRight;
13
14 /*******************************/
15 /* Process left channel sample */
16 /*******************************/
17 *outputLeft = inputLeft;
18 }
```

```
/*
* UTIL.H
*/

#ifndef UTIL_H
#define UTIL_H

short ccan(short M, short *a, short *b, short *w, short **p, short x);
void cdelay(short D, short *w, short **p);
short cfir(short M, short *h, short *w, short **p, short x);
int cround(int a);
short impulse();
short lookup_wave(short L, const short *table, int delta, short *intOffset, unsigned
    short *UfracOffset);
short lookup_waveIntDelta(short L, const short *table, short delta, short *offset);
short tap(short M, short *w, short *p, short i);
short tdl(short M, short *w, short **p, short N, short *gain, short *delta, short x);
void wrap(short M, short *w, short **p);

#endif
```

```
/*
* CCAN.C
*
* Purpose: This code implements an IIR filter in canonical form (DFII) with a
*    circular buffer as described in [1].
*
* Input Arguments:
*    Name: M
*    Description: array order, i.e. array length = M + 1
*
*    Name: *a
*    Description: pointer to feedback coeffients
*
*    Name: *b
*    Description: pointer to feedforward coeffients
*
*    Name: *w
*    Description: pointer to filter states
*
*    Name: **p
*    Description: pointer to oldest filter state (called by reference)
*
*    Name: x
*    Description: input sample
*
* Output Argument:
*    Name: y
*    Description: output sample
*/

#include "util.h"

short ccan(short M, short *a, short *b, short *w, short **p, short x)
{
    short i;
    int y=0, w0;

    **p = x;              /* read input sample in (temporarily stored as **p) */

    w0 = *(*p)++ <<15;    /* begin calculation of w0 by initializing with x */
    wrap(M, w, p);        /* p -> w1 */
```

```
    /* compute new filter state, w0 = -\sum_{i=1}^{M} a_k * w_k */
    for(a++, i=1; i<=M; i++) {
        w0 -= (*a++) * (*(*p)++);
        wrap(M, w, p);
    }
    **p = cround(w0)>>15;    /* round and put w0 into state queue, p -> w0 */

    /* compute filter output, y = \sum_{i=0}^{M} b_k * w_k */
    for(i=0; i<=M; i++) {
        y += (*b++) * (*(*p)++);
        wrap(M, w, p);
    }

    /* update filter states, i.e. wk(n+1) = w{k-1}(n) */
    cdelay(M, w, p);    /* p -> wm */

    return cround(y)>>15;    /* round output and back to 16 bits */
}
```

```
/*
* CDELAY.C
*
* Purpose: This code implements a D-fold delay using a circular buffer as
*    described in [1] for a *fixed-point DSP*.
*
* Input Arguments:
*    Name: D
*    Description: array order, i.e. array length = D + 1
*
*    Name: *w
*    Description: pointer to base address of array
*
*    Name: **p
*    Description: pointer to array (called by reference)
*
*/

#include "util.h"

void cdelay(short D, short *w, short **p)
{
    (*p)--;            /* decrement pointer and... */
    wrap(D, w, p);     /* ...wrap modulo-(D+1) */
}
```

```
/*
* CFIR.C
*
* Purpose: This code implements an FIR filter with a circular delay-line buffer
*    as described in [1] for a *fixed-point DSP*.
*
* Input Arguments:
*    Name: M
*    Description: filter order, i.e. length = M + 1
*
*    Name: *h
*    Description: pointer to filter coeffients
*
*    Name: *w
*    Description: pointer to filter states
*
*    Name: **p
```

```
*    Description: pointer to oldest filter state (called by reference)
*
*    Name: x
*    Description: input sample
*
* Output Argument:
*    Name: y
*    Description: output sample
*/

#include "util.h"

short cfir(short M, short *h, short *w, short **p, short x)
{
    short i;
    int y=0;

    **p = x;    /* read input sample */

    /* compute filter output, y = \sum_{i=0}^{M}h_k * w_k */
    for(i=0; i<=M; i++) {
        y += (*h++) * (*(*p)++);
        wrap(M, w, p);
    }

    /* update filter states, i.e. wk(n+1) = w{k-1}(n) */
    cdelay(M, w, p);    /* p -> wm */

    return cround(y)>>15;    /* round output and back to 16 bits */

}
```