NAME: _____ GRADE: _____

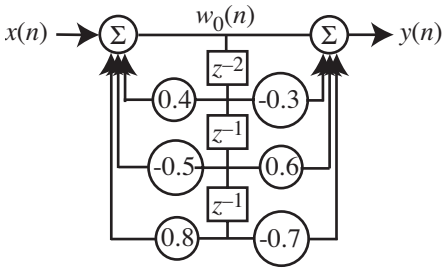
EE442 / EE592 Real-Time Digital Signal Processing
Quiz #4
Open Books, Notes, Calculators (no programs, no graphing)

Each question is worth 10 points unless otherwise noted.

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. Determnine the state equations for the following IIR filter. Do not be concerned about the stability of this filter.



Write a Motorola 56300 assembly code for the following *non-real-time* programming tasks (Questions 3 – 4). The first programming task is done as an example.

3. Write a code which computes the following inner product

$$y = \mathbf{h}^T\mathbf{x} = \begin{bmatrix}0.4 & 0.3\end{bmatrix}\begin{bmatrix}0.2\\0.1\end{bmatrix}$$

Solution:

```
org   p:$100
move #0.4,x0
mpy #0.2,x0,a      ;a = 0.4*0.2
move #0.3,x0
mac #0.1,x0,a      ;a = 0.4*0.2 + 0.3*0.1 (ANSWER IN a)
```

4. For the data byte stored in bit positions 0 – 9 of x0, write a code which computes the odd parity bit and stores this bit in position 10 of x0.

Write a Motorola 56300 assembly code for the following *real-time* programming tasks (Questions 5 – 6), assuming the Modified Pass Pack beginning on p. 307 of the text. Identify by file name (PASS.ASM, PASS.DAT, PROGINIT.ASM, and PROCSTER.ASM) and line number where you would place your instructions. The first programming task is done as an example.

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

Solution:

In PASS.DAT insert at line 11 the following line:

```
g   equ   0.7
```

In PASS.ASM insert at line 83 the following line:

```
   move   #g,x1   ; move g to x1
```

In PASS.ASM replace lines 98 – 100 with the following lines:

```
   mpyr   x0,x1,a ; multiply right sample by 0.7, rounded result in a
   move   a,x0    ; move result to x0 for move to TX_BUFF_BASE in loop_1
```

6. Write a code which FIR filters the left channel input and monitors the output. When the output is less than or equal to –0.5, the code halts (jmp *). Assume the following line is appropriately placed in PASS.DAT:

```
coefs dsm   4           ;FIR filter coefficients
      org x:coefs
      dc    0.1,0.2,0.3,0.4
```

Solution:

Write a C code for the following *real-time* programming tasks (Questions 7 – 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.

7. 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;
```

8. Using the ccan() routine, write code which implements your state equations in Problem 2 for the right channel. The ccan() routine is listed at the end of this quiz.

Solution:

This page for EE592 students only.

9. Assuming the TI dsk_app passcode, write a C code [using the ccan() routine listed at the end of this quiz], which implements the comb filter state equations on p. 167. Identify by file name (user_data.h, initialize_program.c, process_signal.c) and line number where you would place your code. Printouts of the files are listed at the end of this quiz.

Also, assume the following in user_data.h

```
#define CFD 5                       /* comb filter order */
#define CFG 29491                   /* comb filter gain */
extern short CF[CFD+1];             /* array for filter states */
extern short *CF_OldestStatePtr;    /* pointer to oldest state */
```

[CFD equals *m* in Fig. 5.8(a) on p. 168]

Finally, you may reuse as much of your solution to Problem 8 as you wish.

Routines from the TI dsk_app passcode

```
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 8
*outputLeft)
8  {
9      /*******************************/
10     /* Process right channel sample */
11     /*******************************/
12     *outputRight = inputRight;
13
14     /*******************************/
15     /* Process left channel sample */
16     /*******************************/
17     *outputLeft = inputLeft;
18  }
```

```
/*
* 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 */
}
```