1 Lecture Outline

Reading: Chapter 7 Digital Filter Realizations

- C Implementation of IIR Digital Filters
- C Implementation of Sample-by-Sample Processing

2 IIR Program

The following routine ccan.c implements the canonical realization of Fig. 7.2.4 using circular buffers, and replaces can. For simplicity, we assume that the numerator and denominator polynomials have the same order M.

Figure 1: Orfanidis p. 274 Figure 7.2.4. canonical realization form of Mth order IIR filter.

Figure 2: Orfanidis p. 274 samlpe-by-sample filtering algorithm.

ccan()

```
1  float ccan(int M, float *a, float *b, float *w, float **p, float x)
2  {
3      int i;
4      float y=0.0, w0;
5
6      **p = x;          // read input sample in (temporarily stored as **p)
7
8      w0 = *(*p)++;     // begin calculation of w0 by initializing with x
9      wrap(M, w, p);    // p -> w1
10
11     // compute new filter state, w0 = -\sum_{i=1}^{M} a_k w_k
12     for (a++, i=1; i<=M; i++)
13     {
14         w0 -= (*a++) * (*(*p)++);
15         wrap(M, w, p);
16     }
17     **p = w0;    // put w0 into state vector, p -> w0
18
19     // compute filter output, y = \sum_{i=0}^{M} b_k w_k
20     for (i=0; i<=M; i++)
21     {
```

```
22          y += (*b++) * (*(*p)++);
23          wrap(M, w, p);
24      }
25
26      // update filter states, i.e. wk(n+1) = w{k-1}(n)
27      cdelay(M, w, p);      // p -> wm
28
29      return y;    // output sample
30  }
```

3 C Implementation of IIR Filter

In this example, we will read an input text file containing interleaved right/left samples from a stereo audio signal. These samples will be filtered (processed) by a simple two coefficient FIR filter. The output samples will be written to an output text file. The file I/O can be replaced with code which reads and writes samples to an audio device such as a soundcard.

main.c

```
1   #include <stdio.h>
2   #include <errno.h>
3   #include <stdlib.h>
4
5   #include "user_data.h"
6
7   int main (void)
8   {
9       float inputRightSample, inputLeftSample, outputRightSample, outputLeftSample;
10      FILE *inputFilePtr, *outputFilePtr;
11
12      inputFilePtr = fopen("input.txt","r");  // open input file for reading
13      if(inputFilePtr == NULL)     // make sure the input file exists...
14      {
15          perror("Error");     // if not message...
16          exit(1);     // and exit
17      }
18      outputFilePtr = fopen("output.txt","w");     // open output file for writing
19
20      /*****************/
21      /* Initialization */
22      /*****************/
23      initialize_program();
24
25      /*****************************************************************************/
26      /* Main loop - process right, left input samples; get right, left output samples */
27      /*****************************************************************************/
28      while (fscanf(inputFilePtr, "%f%f", &inputRightSample, &inputLeftSample) != EOF)
29      {
30          process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
31          fprintf(outputFilePtr, "%f\n%f\n", outputRightSample, outputLeftSample);
32      }
33
34      fclose(inputFilePtr);   // close input file
35      fclose(outputFilePtr);  // close output file
36
37      return 0;
38  }
```

user_data.h

```
1   #ifndef USER_DATA
2   #define USER_DATA
3
4   void initialize_program(void);
5   void process_signal(float inputRight, float inputLeft, float *outputRight, float *outputLeft);
6
7   #define FILTERORDER 2
8
9   // extern variables are seen by *all* functions in *all* source files, i.e. global variables
```

```
10 // initialization of extern variables in initialize_program.c
11 extern float a_Coeffs[FILTERORDER+1];    // feedback coeffs
12 extern float b_Coeffs[FILTERORDER+1];    // feedforward coeffs
13 extern float states[FILTERORDER+1];      // filter states
14 extern float *oldestStatePtr;            // oldest state pointer
15
16 #include "util.h"
17
18 #endif
```

initialize_program.c

```
1  #include "user_data.h"
2
3  /***************************************/
4  /* Global variable initializations here */
5  /***************************************/
6  float a_Coeffs[FILTERORDER+1]={1.0, 0.0, 0.171573};           // IIR filter feedback coeffs
7  float b_Coeffs[FILTERORDER+1]={0.292893, 0.585786, 0.292893}; // IIR filter feedforward coeffs
8  float states[FILTERORDER+1]={0};                              // filter states all initialized to zero
9  float *oldestStatePtr=states;                                 // oldest state pointer
10
11 void initialize_program(void)
12 {
13
14 }
```

process_signal.c

```
1  #include "user_data.h"
2
3  void process_signal(float inputRight, float inputLeft, float *outputRight, float *outputLeft)
4  {
5      /*******************************/
6      /* Process right channel sample */
7      /*******************************/
8      *outputRight = ccan(FILTERORDER, a_Coeffs, b_Coeffs, states, &oldestStatePtr, inputRight);
9
10     /*******************************/
11     /* Pass thru left channel sample */
12     /*******************************/
13     *outputLeft = inputLeft;
14 }
```

util.h

```
1  #ifndef UTIL_H
2  #define UTIL_H
3
4  // function prototypes
5  float ccan(int M, float *a, float *b, float *w, float **p, float x);
6  void cdelay(int D, float *w, float **p);
7  float cfir(int M, float *h, float *w, float **p, float x);
8  float impulse();
9  float plain(int D, float *w, float **p, float a, float x);
10 float tap(int M, float *w, float *p, int i);
11 float tdl(int M, float *w, float **p, int N, float *gain, int *delta, float x);
12 void wrap(int M, float *w, float **p);
13
14 #endif
```