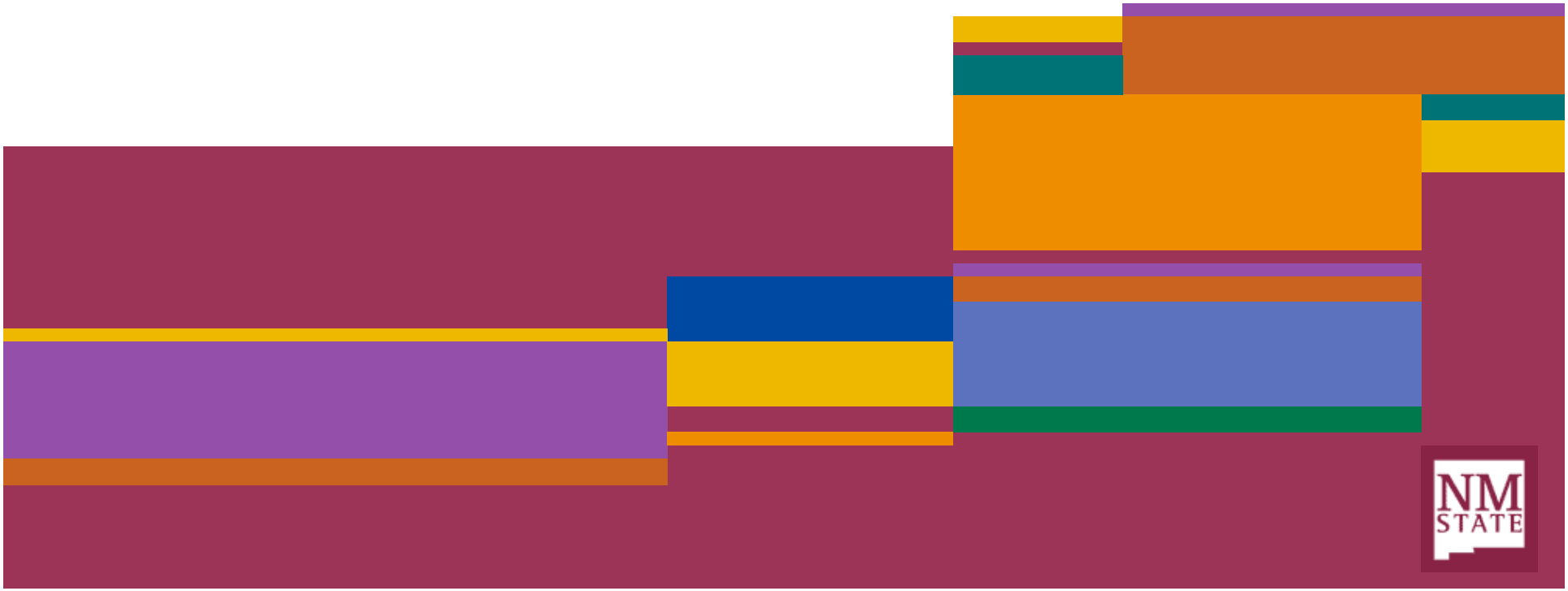
Fixed-Point DSP C Functions



DSP Function

- Based on Orphanidis DSP textbook, we write a collection of DSP functions for fixed-point hardware These include:
  - wrap.c -- pointer wrapping tool for circular queues
  - cdelay.c – tool for pointing to the “new” oldest state in circular delay
  - cfir.c -- FIR filter using circular queue for input samples
  - ccan.c -- IIR filter (canonical form) using circular queue for filter states
- These functions are collected in the folder DSPFunctionsFixedPoint within the DSK_APP

wrap.c

- wrap() (p. 172 Orphanidis) ensures a pointer to circular queue never goes beyond bounds of array
  - M is order of the array, i.e. length of queue is M + 1
  - *w is a pointer to base address of array
  - *p is a pointer into array [since *p is modified by wrap() it must be passed by reference, i.e. a pointer to a pointer **p]
- wrap() circularly wraps pointer p relative to array w by determining if *p has gone past end of array (or before beginning of array) and wraps it modulo *M*
- Since C modulo operator can't wrap for negative values we take this account into account with second IF statement

wrap.c

```
void wrap(short M, short *w, short **p)
{
   if (*p < w || *p > (w + M))
      *p = w + (*p - w) % (M + 1);

   if (*p - w < 0)
      *p += M+1; // stupid C modulus operator
}
```

cdelay.c

- cdelay() (p. 177 Orphanidis) will point to new, oldest state
  - D is order of array, i.e. length of array is D + 1
  - *w is a pointer to base address of array
  - *p is a pointer into array (p is an address). Since *p is modified by cdelay () it must be passed by reference, i.e. a pointer to a pointer **p
- Since we want *p to point to oldest sample or state, this routine simply decrements pointer to new, oldest state and wraps pointer back around for circular addressing

cdelay.c

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

cfir.c

- cfir() (p. 173 Orphanidis) implements convolution assuming input samples are stored in a circular queue
  - M is filter order, i.e. length of filter is M + 1
  - *h is a pointer to base address of filter coefficients
  - *w is a pointer to base address of input queue
  - *p is a pointer to oldest sample in input queue passed by reference, i.e. **p
  - x is newest sample

cfir.c

- Routine first blows away oldest sample with newest, i.e. **p = x so that *p points to newest sample.
- Next we accumulate a sum of products between filter coefficients and appropriate past input sample
  - Each time through loop we advance pointer to coefficients and pointer to input samples
  - Each time input sample pointer is incremented we must wrap pointer
  - Accumulation of products is stored as int y
- Next, with cdelay() we point to new, oldest sample
- Finally, we round y and divide by 2^{15} so that it contains proper output value and return this output sample

cfir.c

```
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 */
}
```

ccan.c

- ccan() (p. 302 Orphanidis) implementents DFII (canonical form) assuming filter states are stored in a circular queue
  - M is filter order, i.e. length of filter state vector is M + 1
  - *a is a pointer to base address of feedback coefficients
  - *b is a pointer to base address of feedforward coefficients
  - *w is a pointer to base address of filter states
  - *p is a pointer to oldest state in filter states queue passed by reference, i.e. **p
  - x is newest sample
- Compute new filter state, filter output, & update states
  - $w_0(n) = x(n) - a_1 w_1(n) - a_2 w_2(n) - \ldots - a_M w_M(n)$
  - $y(n) = b_0 w_0(n) + b_1 w_1(n) + \ldots + b_M w_M(n)$
  - $w_k(n+1) = w_{k-1}(n)$

ccan.c

- Create an int variable, w0 for new filter state
  - w0 is initialized with input sample (multiplied up for int) and we point to first filter state (wrapping if necessary)
- First ‘for’ loop computes negative sum of products between feedback coefficients and filter states (wrapping if necessary)
  - After sum of products is completed, divide by 2^{15} for storage of new filter state as short
- Second ‘for’ loop accumulates sum of products between feedforward coefficients and filter states (wrapping if necessary)

ccan.c

- Third, back pointer up and wrap with cdelay() to point to new, oldest state
- Round y and divide by 2^{15} so that it contains proper output value (return this output sample)

ccan.c

```
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 */
}
```

util.h

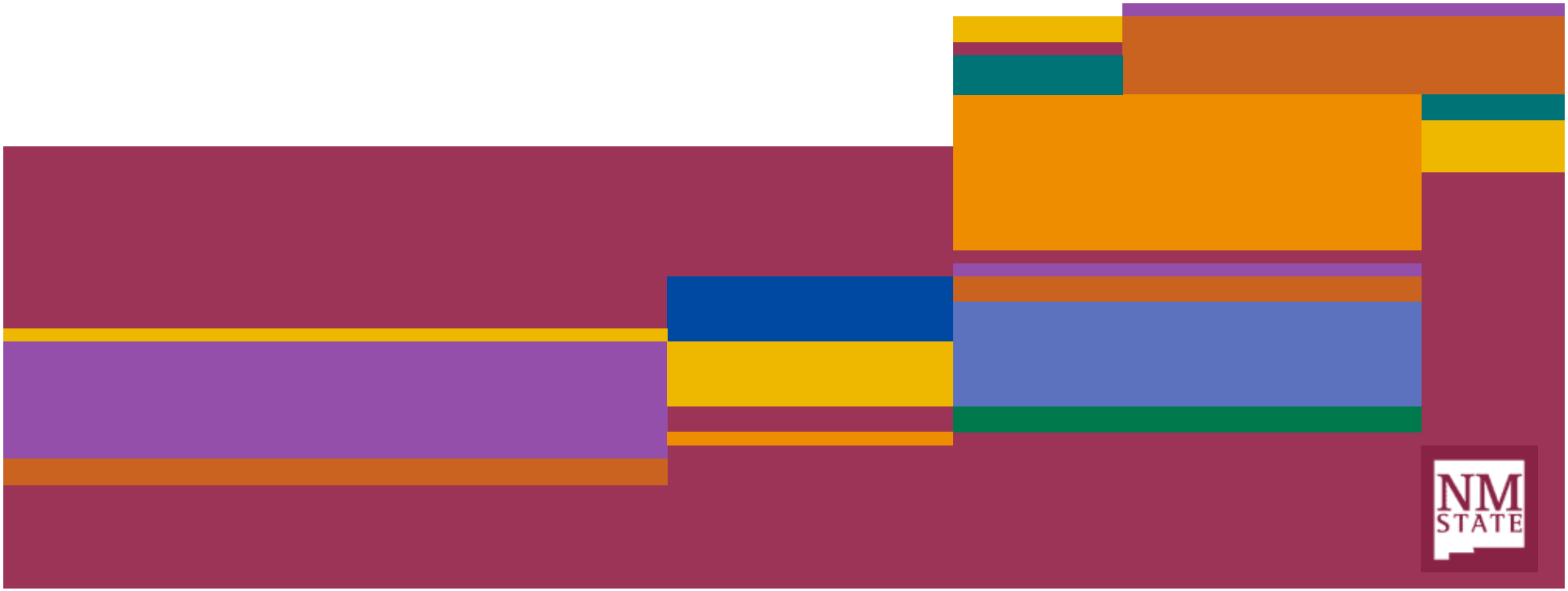
- Since these DSP functions are useful and will be reused, we put them into a utility library to make their reuse easy
- The DSPFunctionsFixedPoint library has the prototypes

```
#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);
void wrap(short M, short *w, short **p);

#endif
```

Real-Time FIR Filtering



Real-Time FIR Filtering

- In DSK_APP, we must edit several files
  - user_data.h (like pass.dat)
  - initialize_program.c (like proginit.asm)
  - process_signal.c (like procster.asm)

user_data.h

```
#ifndef USER_DATA
#define USER_DATA

void initialize_program();
void process_signal(short inputRight, short inputLeft,
short *outputRight, short *outputLeft);

#define FILTERORDER 1

extern short coeffs[]; /* FIR filter coefficients */
extern short states[]; /* filter states */
extern short *oldestStatePtr; /* oldest state pointer */

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

extern = variables that will be defined later

initialize_program.c

```
#include "user_data.h"

/***************************************/
/* Global variable initializations here */
/***************************************/
short coeffs[FILTERORDER+1]={16384,16384};     /* FIR coefs */
short states[FILTERORDER+1];      /* filter states */
short *oldestStatePtr;            /* oldest state pointer */

void initialize_program()
{
    short i;

    oldestStatePtr = states;      /* setup ptr to states, clear
*/
    for(i=0;i<=FILTERORDER;i++)
        states[i] = 0;
}
```

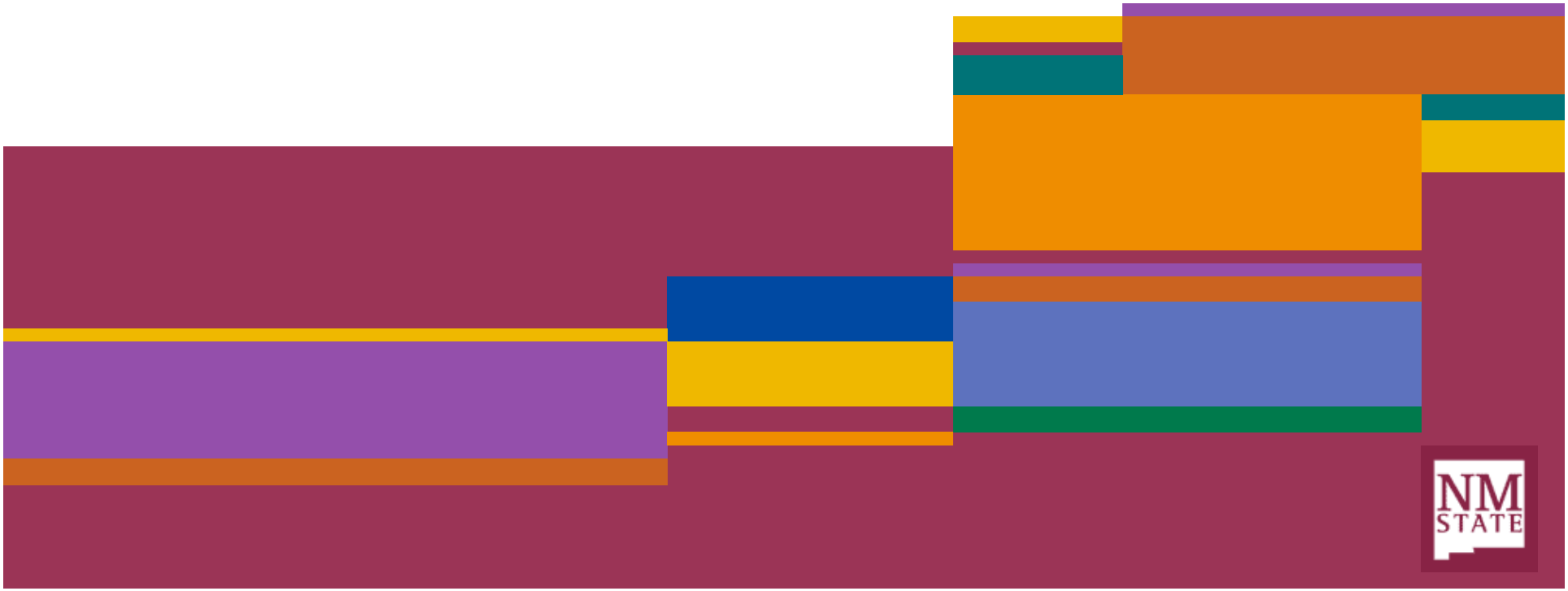
process_signal.c

```
void process_signal(short inputRight, short inputLeft, short
*outputRight, short *outputLeft)
{

    /*********************************/
    /* Process right channel sample */
    /*********************************/
    *outputRight = cfir(1, coeffs, states, &oldestStatePtr, inputRight);

    /*********************************/
    /* Process left channel sample */
    /*********************************/
    *outputLeft = inputLeft;
}
```

Real-Time IIR Filtering



Real-Time IIR Filtering

- In DSK_APP, we have several functions that must be properly coded
  - user_data.h
  - initialize_program.c
  - process_signal.c

Real-Time IIR Filtering (user_data.h)

```
#ifndef USER_DATA
#define USER_DATA

void initialize_program();
void process_signal(short inputRight, short inputLeft, short
*outputRight, short *outputLeft);

#define FILTERORDER 2

extern short a_Coeffs[];          /* feedback coeffs */
extern short b_Coeffs[];          /* feedforward coeffs */
extern short states[];            /* filter states */
extern short *oldestStatePtr;     /* oldest state pointer */

#include "util.h"

#endif
```

Real-Time IIR (initialize_program.c)

```
#include "user_data.h"

/*****************************************/
/* Global variable initializations here */
/*****************************************/
short a_Coeffs[FILTERORDER+1]={32767, 0, 5622};   /* feedback coeffs */
short b_Coeffs[FILTERORDER+1]={9598, 19195, 9598};/* feedforward coeffs *
short states[FILTERORDER+1];  /* filter states */
short *oldestStatePtr; /* oldest state pointer */

void initialize_program()
{
    short i;

    oldestStatePtr = states; /* setup pointer to states and clear it out
    for(i=0;i<=FILTERORDER;i++)
        states[i] = 0;
}
```

Real-Time IIR Filtering (process_signal.c)

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short
*outputRight, short *outputLeft)
{
    /*********************************/
    /* Process right channel sample */
    /*********************************/
    *outputRight = ccan(FILTERORDER, a_Coeffs, b_Coeffs, states,
&oldestStatePtr, inputRight);

    /*********************************/
    /* Process left channel sample */
    /*********************************/
    *outputLeft = inputLeft;
}
```

Other C Codes

- DeLeon has written a collection of fixed- and floating-point pass codes (both mono and stereo) which call process_signal() in the same way DSK_APP does.
  - These codes receive/send right and left input/output samples, from/to a file rather than a codec
- These codes allow one to develop signal processing routines on their platform/IDE of choice (for me Macintosh/Xcode) and simply add the routines to the CCS DSK_APP project for compilation and real-time evaluation

main.c (generic passcode)

```
#include <stdio.h>
#include "user_data.h"

int main (void)
{
   short inputRightSample, inputLeftSample, outputRightSample, outputLeftSample;
   FILE *fopen(), *inputFilePtr, *outputFilePtr;

   inputFilePtr = fopen("input.txt","r");         /* open input file for reading */
   outputFilePtr = fopen("output.txt","w");       /* open output file for writing */

   /******************/
   /* Initialization */
   /******************/
   initialize_program();

   /*****************************************************************************/
   /* Main loop - process right, left input samples; get right, left output samples */
   /*****************************************************************************/
   while (fscanf(inputFilePtr, "%hd%hd", &inputRightSample, &inputLeftSample) != EOF) {
      process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
      fprintf(outputFilePtr, "%hd\n%hd\n", outputRightSample, outputLeftSample);
   }

   fclose(inputFilePtr);   /* close input file */
   fclose(outputFilePtr);  /* close output file */

   return 0;
}
```