DSP and Communications Lab

Real-Time Block Design and Simulation on DSP Board

JACOBS UNIVERSITY

Real-Time Block Design

Objectives

- Learn how to program real-time DSP blocks in C
- Implement/test simple delay block
- Implement/test FIR block

Structured DSP Programming in C

One Method

- Just think of algorithm as sequential program
- Code up the different steps
- Problems: Not so structured. Can be confusing. Easy to make mistakes.

Structured Method

- Like what you did in MATLAB blocks in last lab
- Simplifies process and keeps it clean
  - Draw the block diagram
  - Code/test the individual blocks
  - Connect the blocks together

Functions of a single DSP Block

[blockname]_init(params...)

- Initializes the block
- Allocates space for any buffers / state variables
- Returns a pointer to the initialized state structure

[blockname](state *, in_buffer, out_buffer)

- Function that processes samples
- Takes samples from in_buffer and fills out_buffer
- Can also modify state as appropriate

[blockname]_modify()

- Allows you to change parameters or state during runtime

Example: Delay Block (delay.h header)

delay.h

- “Header” file
- Declares functions prototypes and constant values
- Can be “included” in other files that use those functions and values

Some preliminaries

- Comments

```
// This is a comment
/* This is also
   a comment! */
```

- Including a header only once!

```
#ifndef _delay_h_
#define _delay_h_

. . . Header declarations . . .

#endif /* _delay_h_ */
```

Example: Delay Block (delay.h)

#define statements

- Preprocessor directive
- **#define *name value***
- *name* get substituted with *value* everywhere in the code at compile time
- An efficient way to implement a “constant”
- Does not create any additional code

What is the difference of the following 2 statements?

```
#define NAME value

int NAME = value;
```

Example: Delay Block (delay.h)

#define statements

```
#define NAME value

/* Size of buffer (samples). Controls maximum delay. */
#define DELAY_BUFFER_SIZE 16384

/* Mask. Used to implement circular buffer */
#define DELAY_BUFFER_CMASK (DELAY_BUFFER_SIZE-1)

/* Which memory segment the data should get stored in */
//#define DELAY_SEG_ID 0 /* IDRAM - fastest, but smallest */
#define DELAY_SEG_ID 1 /* SRAM - a bit slower, but bigger */

/* Allows alignment of buffer on specified boundary. */
#define DELAY_BUFFER_ALIGN 128

/* Samples required for 1MS of Delay */
#define DSP_SAMPLES_PER_SEC 8000
#define DELAY_SAMPLES_1MS (DSP_SAMPLES_PER_SEC/1000)
```

Example: Delay Block (delay.h)

State Structure

```
typedef struct
{
    float buffer[DELAY_BUFFER_SIZE];
    unsigned int del;
    unsigned int h, t;
} delay_state_def;
```

Similar to what you did in MATLAB

Buffer of samples

Current delay value

Head/Tail pointers

Example: Delay Block (delay.h)

Function “prototypes”

```
/*-- Function Prototypes -------------------------------------------------------*/

/* Initializes the delay block */
delay_state_def *delay_init();

/* Change delay parameters */
void delay_modify(delay_state_def *s, unsigned int new_delay);

/* Processes a buffer of samples for the delay block */
void delay(delay_state_def *s, const float x_in[], float y_out[]);
```

Defines the “calling convention” but not the actual functions

Must be “included” in each file that uses the respective functions

Example: Delay Block (delay.c)

Next, look at **delay.c**

```
#include <std.h>
#include <sys.h>
#include <dev.h>
#include <sio.h>
```

Include the named header files

System header files usually denoted with < >

Consult the TI documentation to see which need to be included

```
#include "delay.h"
#include "dsp_ap.h"
```

These are includes for your design

dsp_ap.h from “golden project” (described later)

Example: Delay Block (delay.c)

Init Function (comment block)

```
/*----------------------------------------------------------------------------
 * delay_init()
 *      This function initializes a delay block with a delay of 0.
 * Inputs:
 *      None.
 * Returns:
 *      0       An error occurred
 *      other   A pointer to a new delay structure
 *-------------------------------------------------------------------------*/
delay_state_def *delay_init()
{
   . . .
```

Strongly recommended you do these comment blocks

Show input/output of functions

Help you document what you are doing and maintain structure

Example: Delay Block (delay.c)

Init Function

```
delay_state_def *delay_init()
{
    delay_state_def *s;

    /* Allocate a new delay_state_def structure.  Holds state and parameters. */
    if ((s = (delay_state_def *)MEM_calloc(DELAY_SEG_ID, sizeof(delay_state_def),
          DELAY_BUFFER_ALIGN)) == NULL)
    {
        SYS_error("Unable to create an input delay floating-point buffer.", SYS_EUSER, 0);
        return(0);
    }

    /* Set initial delay to 0 */
    s->t = 0;
    s->h = 0;

    /* Success.  Return a pointer to the new state structure. */
    return(s);
}
```

Example: Delay Block (delay.c)

Modify Function

```
/*---------------------------------------------------------------------------
 * delay_modify()
 *      Change operating parameters of the delay block.
 * Inputs:
 *      s               A pointer to the delay state structure
 *      new_delay       The new delay value
 *---------------------------------------------------------------------------*/
void delay_modify(delay_state_def *s, unsigned int new_delay)
{
    /* Check the requested delay */
    if (DELAY_BUFFER_SIZE < (new_delay + BUFFER_SAMPLES))
    {
        /* Make delay maximum */
        new_delay = DELAY_BUFFER_SIZE-BUFFER_SAMPLES;
    }

    /* Change the head of the buffer to obtain the requested delay.  Do circular. */
    s->h = /* REPLACE ME!!! */  ;
}
```

Example: Delay Block (delay.c)

Processing Function

```
/*---------------------------------------------------------------------------
 * delay()
 *      Process one buffer of samples with the delay block.
 *-------------------------------------------------------------------------*/
void delay(delay_state_def *s, const float x_in[], float y_out[])
{
    int i;

    /* Read all input samples into tail of buffer */
    for (i=0; i<BUFFER_SAMPLES; i++)
    {
        s->buffer[s->t] = x_in[i];
        s->t++; s->t &= DELAY_BUFFER_CMASK;
    }

    /* Read all output samples from head of buffer */
    for (i=0; i<BUFFER_SAMPLES; i++)
    {
         /* REPLACE ME!!! */
    }
}
```

Note: Defined in dsp_ap.h. It is the number of samples that get processed each time delay() is called

Example: Delay Block (delay.c)

Using the delay block

```
/* Define a pointer to a delay state
structure */
delay_state_def *delay1;

.
.
.

/* Somewhere in the initialization code */
delay1 = delay_init();

/* Set the delay */
delay_modify(delay1,  100);

.
.
.

/* Somewhere in the processing code */
delay(delay1, input_samples,
output_samples);
```

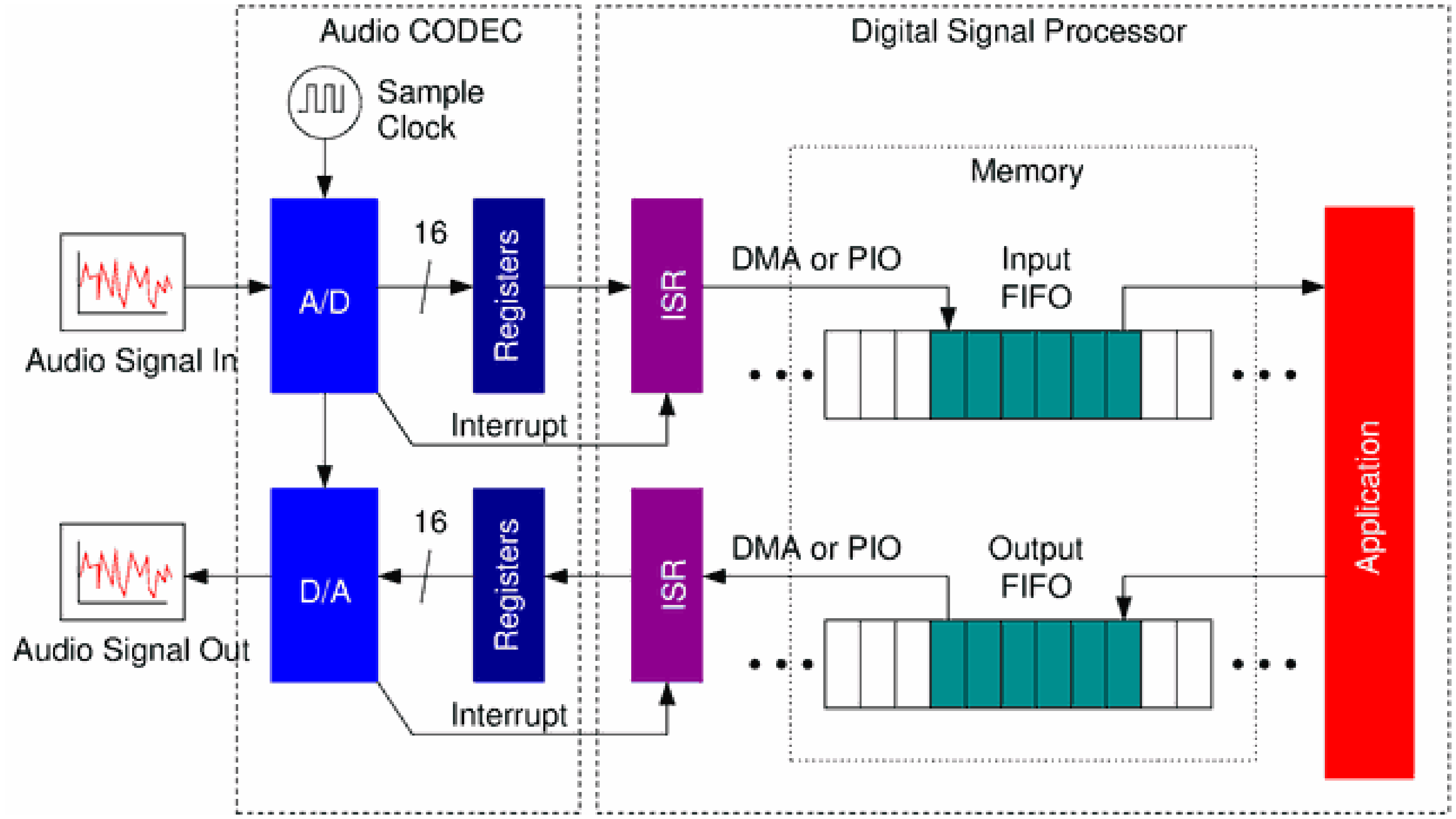
Compare to MATLAB code

(more elegant)

Cascading blocks simple

```
delay(delay1, input_samples,
temp_samples);
delay(delay2, temp_samples,
output_samples);
```

Audio Processing on DSP Boards



Golden Project

Motivation

- Setting up a new project in CCS can be a pain
- Many parameters to set, device drivers to specify
- Much lab time can be wasted tracking down project-related “bugs”

Golden Project

- “Golden” because it should always work
- Implements the “surrounding code” needed for audio processing
- You can mostly just insert your DSP designs in it

Golden Project

Eliminates the need to do the following tasks

- Creating the project
- Creating the DSP/BIOS CDB file and setting options
- Allocating audio codec buffers
- Converting samples to/from floating point

Using the Golden Project

1. Download the right file

golden_prj_c.zip	If you store the project on C:/ like on lab computers
golden_prj_d.zip	If you store it on D:/

2. Unzip it
3. Open dsp_ap.pjt (project file) in CCS
4. Modify files as needed
   - dsp_ap.h
   - dsp_ap.c
   - Note: should leave other files alone (but can create new files!)

Golden Project: dsp_ap.h

Important #defines to consider changing

DSP_SOURCE

sets input samples to come from “line in” (PC) or “mic” (headset)

DSP_SAMPLE_RATE

Sets the sample rate from 8kHz up to 96kHz

Golden Project: dsp_ap.c

Functions you can (should) modify

dsp_init()

Allows you to allocate variables at beginning

Good place to put calls to the blockname_init() functions of all blocks

Return 0 to indicate success, or nonzero to indicate failure

dsp_process()

Called once for each buffer of samples

```
void dsp_process(
        const float inL[ ],
        const float inR[ ],
        float outL[ ],
        float outR[ ])
{
```

Inputs come as floating point: inL, inR

You generate output samples: outL, outR

Golden Project: Debug / Error Indicators

Fatal errors

Rapidly flashing LEDs

Also happens if you return != 0 from dsp_init()

If this occurs, halt board and check the message log (DSP tools)

LED 3

Flashes slowly on/off for each buffer processed

Shows you that things are at least running (samples streaming)

If this doesn’t happen, something got stuck in dsp_process()

Pass-through / loopback mode

Turn DIP switch 3 on (= up)

Copies input samples to output samples (should hear yourself talk)

Should always work if dsp_init() did not fail (dsp_top.c not modified)

Common Pitfalls in DSP Code

Stack usage

Do not declare buffers as local variables!

```
void process()
{
    float my_buffer[16384];
    int i;

    for (i=0; i<16384; i++) { ... }

    ....
}
```

DSP has limited resources

Stack is intended for small temporary variables, not big buffers

Put buffers in state variable (preferred) or as global

Common Pitfalls in DSP Code

Buffer Sizes

Be careful, since there are multiple buffer sizes:

Buffers of input/output samples

Buffer of delayed samples stored in state structure

Buffer of FIR coefficients

Etc.

They can all be different lengths

Critical in **for** loops

Make sure you know what you are doing!

```
float in_buff[BUFFER_SAMPLES];
float
buffer2[DELAY_BUFFER_SAMPLES];

for (i=0; i<DELAY_BUFFER_SAMPLES;
i++)
{
    buffer2[i] = in_buff;
    ....
}
```

Common Pitfalls in DSP Code

Pointers / Arrays

Related to the last point

Be careful not to write/read past the end of an array

You will not see an explicit error

Memory corruption is very difficult to track down

You will just see strange behavior or things will hang

Common Pitfalls in DSP Code

Allocating memory

DSP has limited resources

Have different types of memory (speed/size)

If you allocate a buffer too big in IDRAM (fastest/smallest)
  your program crashes

If this happens, you should see allocation error in LOG

Also, you may need to increase size of heap in DSP BIOS Settings

(TA can help you with this)

Common Pitfalls in DSP Code

Nested Loops

for (i= ...) { for (j= ...) ... }

Make sure you use different variables for inner/outer loops!

This sounds silly, but it happens all the time in student code

Variable types

Use **int** or **unsigned** type for loops and **float** or **double** for samples

Again, sounds silly, but many students are careless with this

Causes weird behavior

Common Pitfalls in DSP Code

Running old code

Make sure you are actually running your latest code on the DSP board!

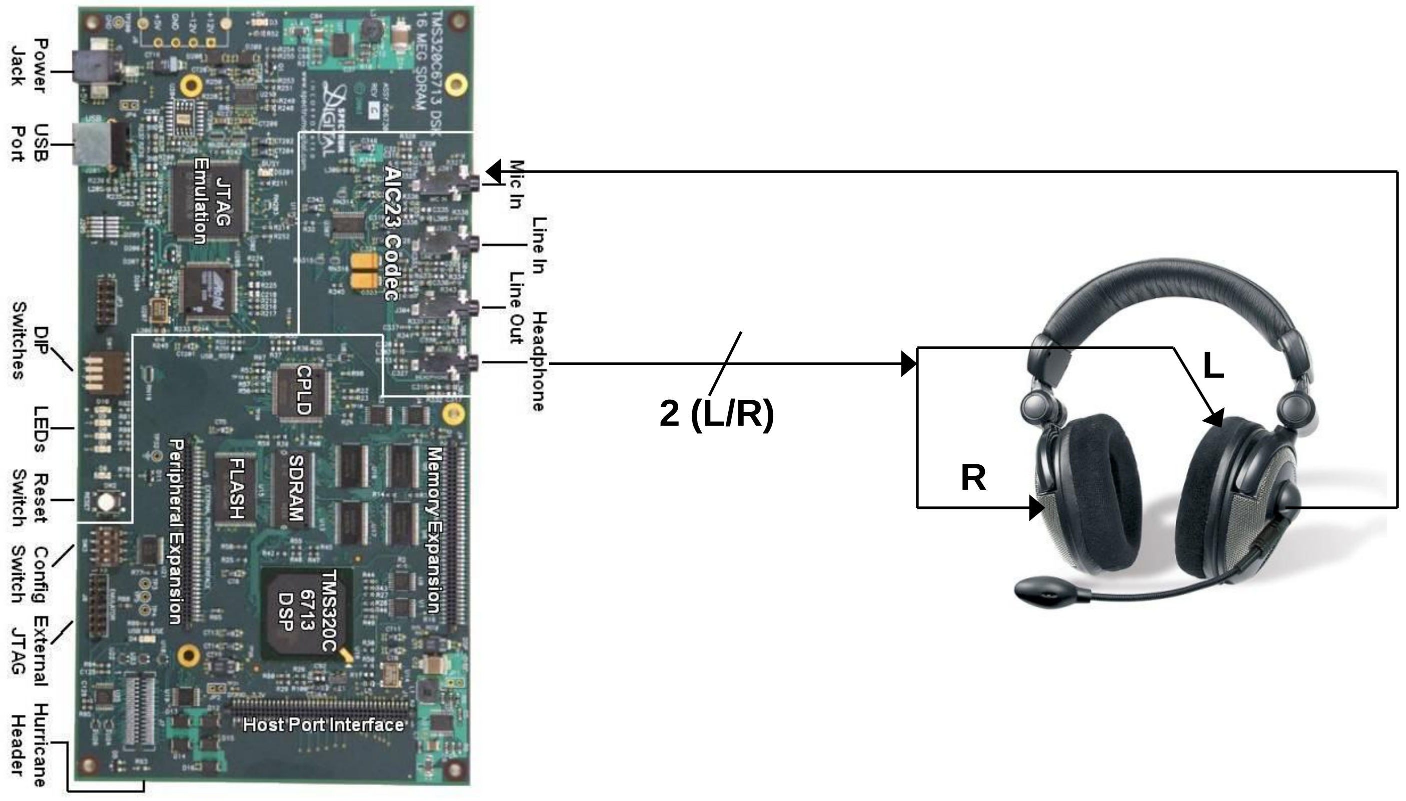
Always explicitly

save
rebuild
(halt), upload to board
run

If you see strange things happening with breakpoints (they don’t hit, or the code stops on a line with no code)
You are not running the version you see on the screen!

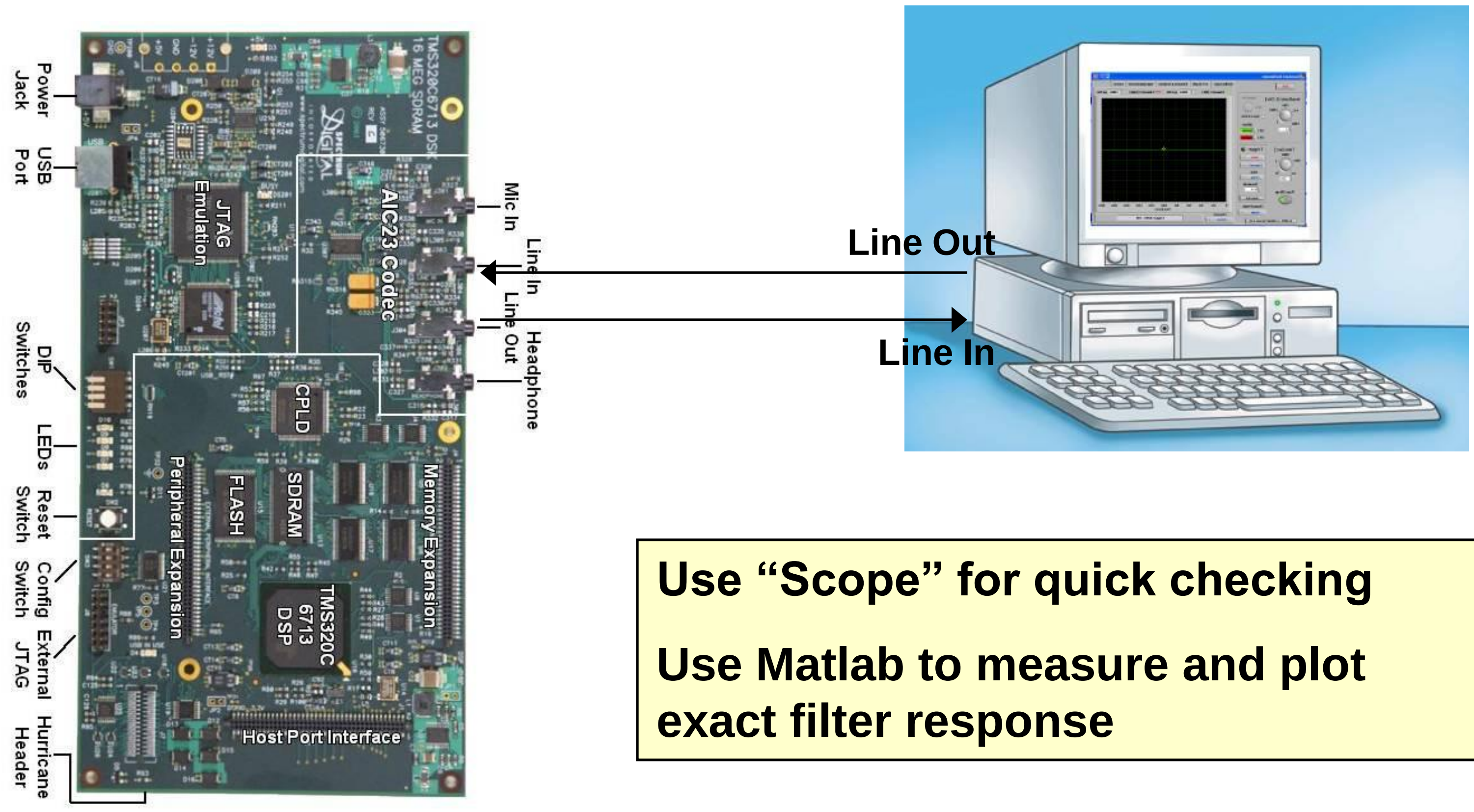
Testing your Design

Using the headset (for delay block)



Testing your Design (2)

Using PC and Scope Program or Matlab (FIR filter)



Use "Scope" for quick checking

Use Matlab to measure and plot exact filter response

Questions?