Chapter 5

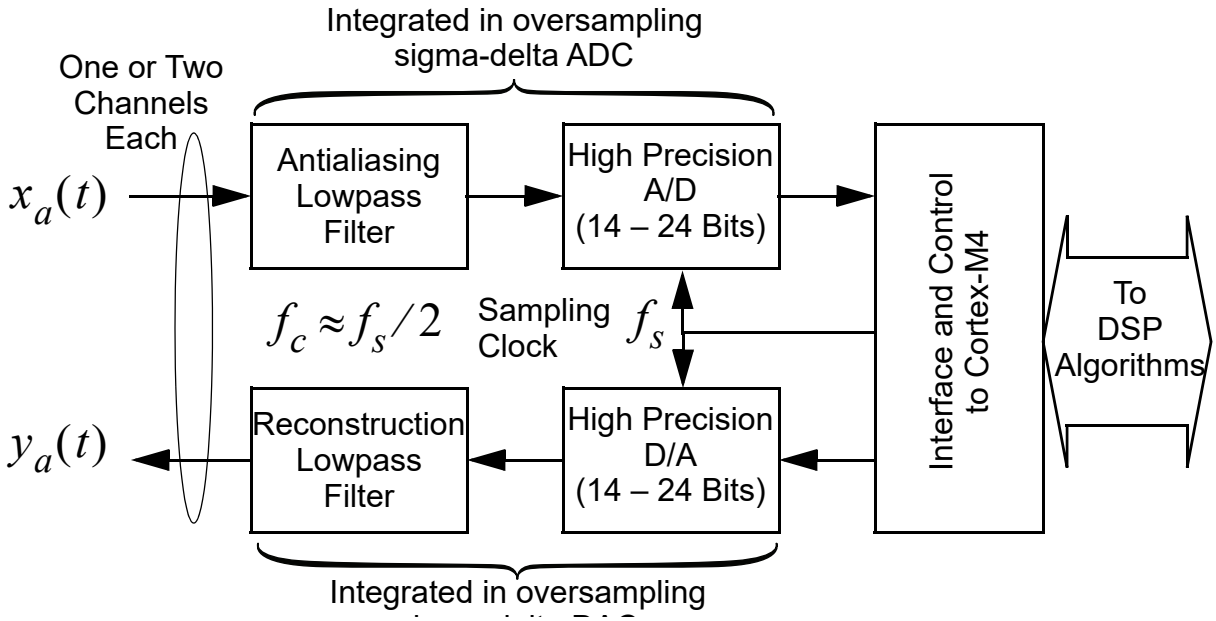
Analog Input and Output

Introduction

We now consider audio interfacing on Cortex-M4 development boards: Cypress FM4 Pioneer, STM32F407 Discovery with Wolfson Pi Audio A/D–D/A PCB, and the LPC4088 Quickstart + baseboard. The first and third boards contain the Wolfson WM8731 audio codec, while the ST board the Wolfson WM5102 codec. The ARM DSP LiB supplies the codec libraries for interfacing to these codecs.

Analog I/O Basics

- An idealized analog interface circuit (AIC) is the following:



- It is common for the AIC to have two dedicated inputs and two dedicated outputs, e.g., a stereo audio codec (CODer/DECoder)
- Since the intent is to process audio signals, the lowpass sampling theory applies, so the analog input must not contain any frequency content above the folding frequency, which in this case is $f_s/2$
  - To prevent aliasing and keep unwanted noise out of the system, an antialiasing filter is typically employed
  - For the case of an ideal anti-aliasing filter, the cutoff frequency is $f_c = f_s/2$
  - The output analog signal as obtained from the DAC is post filtered with a lowpass reconstruction filter to remove spectral translates that exist at frequencies above $f_s/2$
- With an oversampling sigma delta ($\Sigma\Delta$) ADC and DAC the sampling rate is still f_s, but the oversampling rate may be $64f_s$ or higher
- The nice thing about $\Sigma\Delta$ converters is that they incorporate high order anti-aliasing filters (in the ADC) and high-order reconstruction filters (in the DAC)
- Again thinking in terms of an ideal AIC, we would like to have:
  - Software control of the sampling rate, possibly asynchronous sampling rates at the A/D and D/A

  - Software control of the input filter cutoff frequency (low end and high end if a bandpass filter is also available, a selectable dc coupled input for certain control applications
  - Software control of the reconstruction filter cutoff frequency
  - Most stereo audio codecs are at least 16-bits with sampling rates up to 44.1/48 kHz and 96 kHz, or higher supported
  - Lower sampling rates, to support telephony grade audio is also helpful for greater complexity processing needs
  - Sample-by-sample processing or Frame-based processing using DMA for improved efficiency at the expense of increased latency

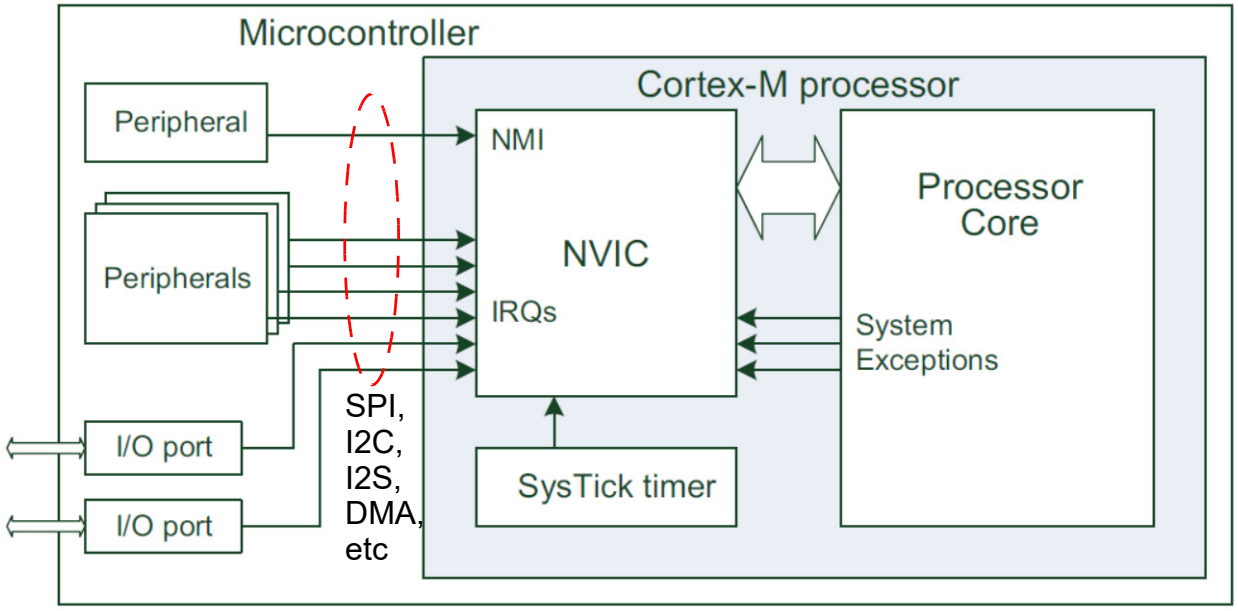
Analog I/O on the Cortex-M4

- For the Cortex-M4 to interface with any form of analog I/O, e.g., ADC and DAC, we need to understand a little bit about interrupt data processing and the nested vector interrupt controller (NVIC)
- From a programming standpoint, the basics of interrupt service routines (ISRs) are to have the CPU respond to the ticking of the sampling clock in such a way that the flow of input samples to output samples never skips a beat
- In some cases timers are used to obtain a programmable sampling rate clock

- An externally supplied sampling rate clock, e.g., a divided down USB–1.1 12.288 MHz crystal oscillator is common on M4 boards such as the FM4 and others

Cortex-M Interrupt Basics

- As the name implies, the processor is halted from its normal processing, and required to execute an ISR
- On the Cortex-M4 the NVIC manages all interrupt requests (IRQs) generally referred to as exceptions



The NVIC responds to various *exception* types

- 1–240 requests are possible, with different priorities assigned to each
- System exceptions are also exist, and are assigned a low exception number
- Within a project the assembly file `startup_xxxx.s` con-

tains the mapping of exception numbers to IRQ handlers:

- See Chapters 7 and 8 of Yiu for more details

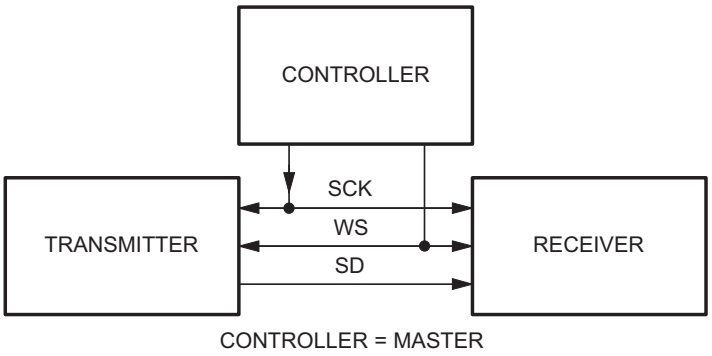
Interrupt Handlers Used in LiB Real-Time DSP

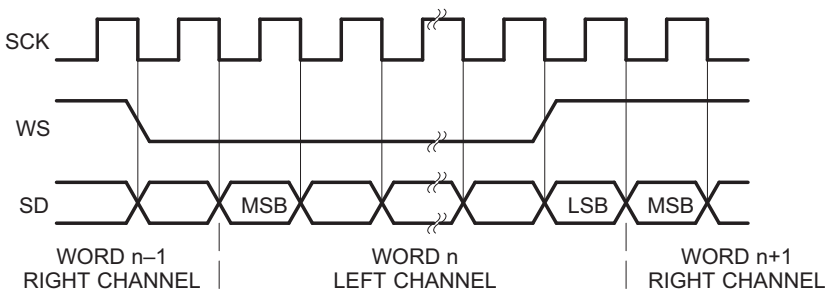
- For the purposes of actually coding real-time DSP there are *action areas* within a typical Keil project where we must pay special attention to time critical coding
- The action area depends upon how the codec is interfaced to the Cortex-M4 and how I/O audio samples are managed as they move between the codec and the Coretex-M4
  - For the STM32F4 this means a serial peripheral interface (SPI), specifically the `SPI2_IRQHandler()`, is talking to the codec and establishing I2S (see below I2S discussion) for sending and receiving audio samples
  - On the STM32F4 the inter-integrated circuit (I2C) bus is used for initialization
  - For the Cypress FM4 and LPC4088 the `I2S_HANDLER()` is used directly in this implementation; again I2S is used to talk with the audio codec

I2S For Audio Communication

- The *Inter-IC Sound* or *Integrated Interchip Sound* bus interface standard (I^2S or I2S) is used for connecting digital audio devices together
  - Originally developed by Phillips (now NXP) in 1986

- The ADC of an audio codec captures audio samples as pulse code modulation (PCM) and similarly the codec DAC receives PCM data for outputting analog signals
- The I2S standard efficiently sends and receives the PCM data between integrated circuits
- Separate clock and serial data signals are employed to insure low clock jitter
- Right and left audio channels are supported at defined bit widths
- At minimum three wires/lines are required:
  - Continuous serial clock (SCK)
  - Word select (WS)
  - Serial data (SD)

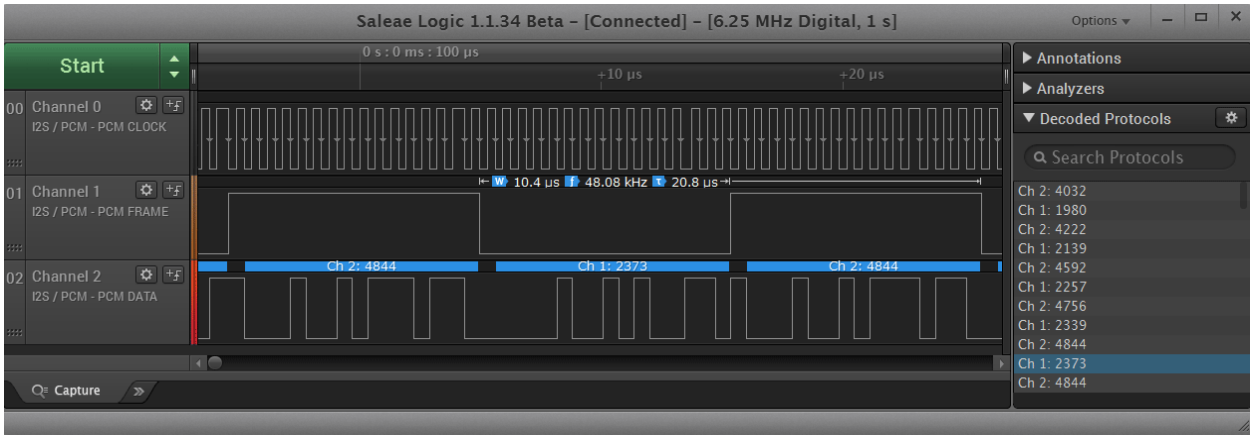




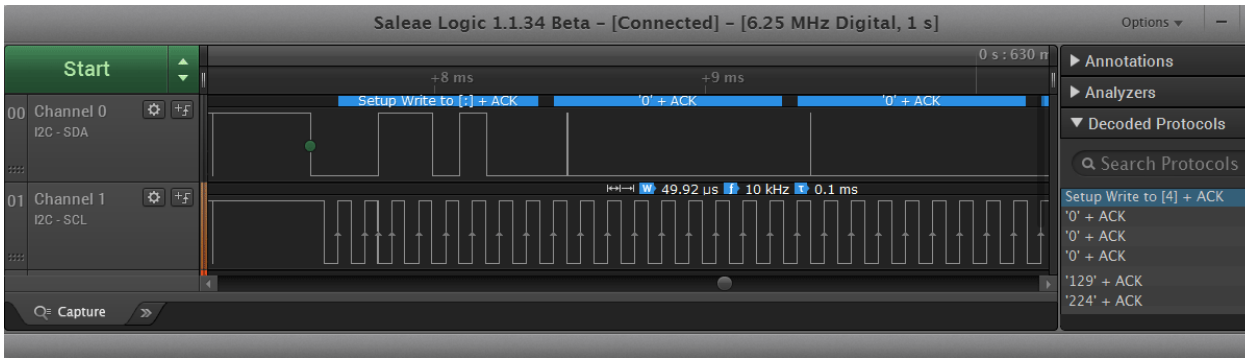
Notice how the Word Select (WS) controls the mode: right channel (high) or left channel (low)

Here we use 16-bits per audio sample on both left and right channels

- On all of the boards, I2S is used with a 16-bit word length
- As an example the Saleae Logic[1] decodes I2S
- I2S decoding on the RX channel (codec point of view) on the STM32F4 from the Pi Audio card:



  - Here the PCM frame clock is the same as Word Select
  - Both channels are processing a 1 kHz sinusoid, but one has twice the gain of the other (GUI slider), so there is a 2:1 difference in values between Ch1 and Ch2
  - Note also the frame rate per channel is ~48 kHz
- FYI, at program start-up the I2C is active:

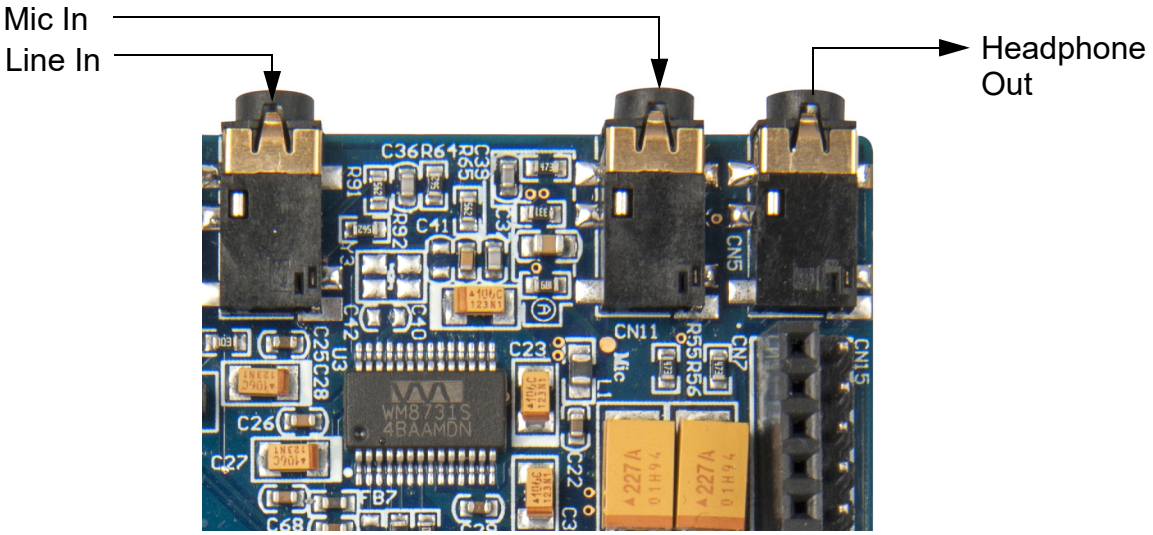


  - **Note**: The I2C clock is 10 kHz

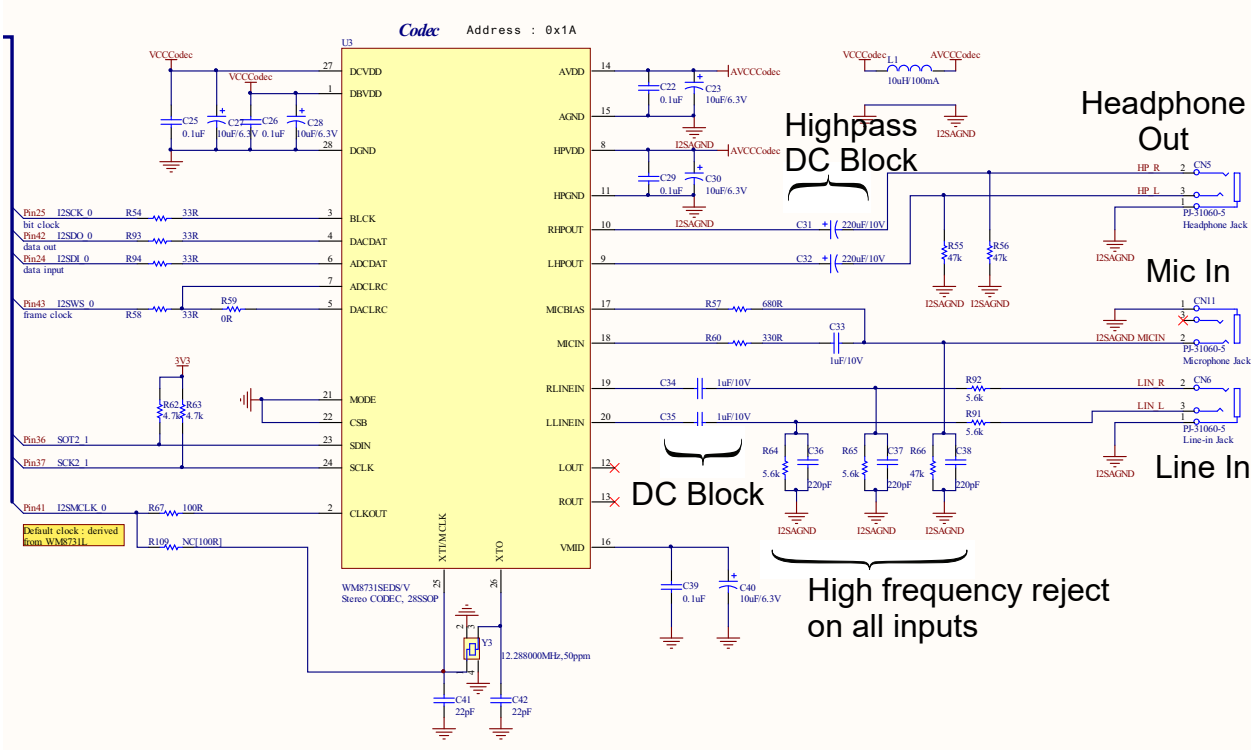
1. https://www.saleae.com

Wolfson WM8731 Codec on Cypress FM4

- We now provide some details about the internals of the Audio codec found on the Cypress FM4 board, in particular the ADC and DAC characteristics

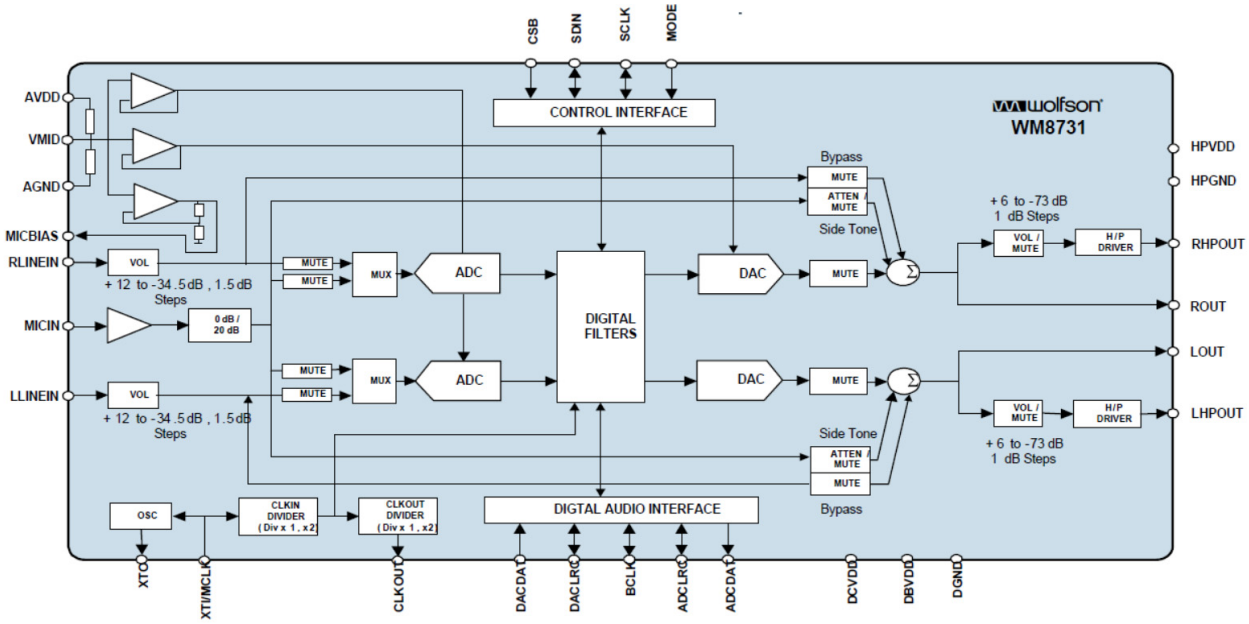


- The FM4 specific circuit interfaces give some insight into additional filtering in the analog domain



Audio Codec Specifics

- The subsystem of interest here is the WM8731 audio codec which has audio interfaces in the upper right of the board
- The device supports sampling rates from 8–96 ksps
- The device supports audio sample bit widths of 16/20/24/32
- The system block diagram is shown below:



- Clearly not as full featured as the WM5102, but adequate for our purposes
- The frequency response specs are given for two over sampling rates 256fs <=> Type 0 digital filtering and 272fs <=> Tyoe 1 filtering
- The Type 0 mode is what we will be using, at least at 48 ksps:

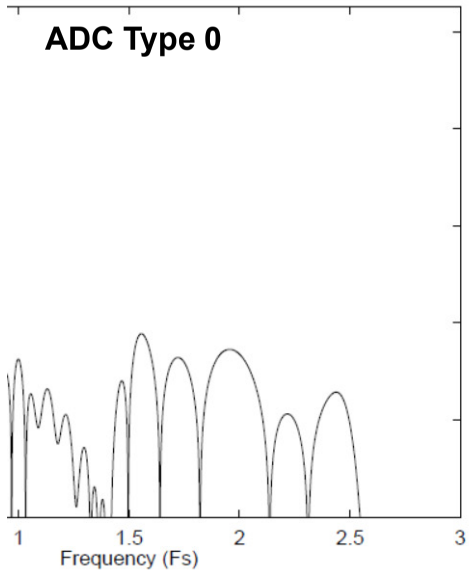
Amplitude Response

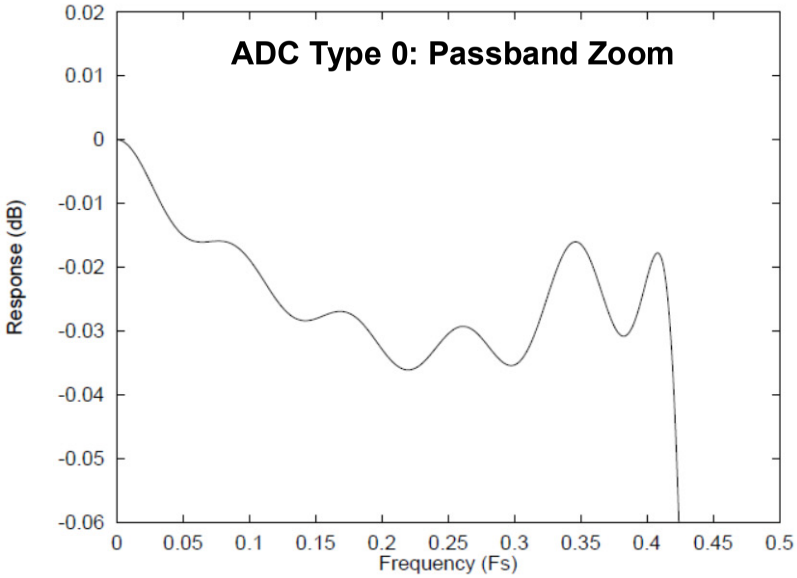
PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
ADC Filter Type 0 (USB Mode, 250fs operation)					
Passband	+/- 0.05dB	0		0.416fs	
	-6dB		0.5fs		
Passband Ripple				+/- 0.05	dB
Stopband		0.584fs			
Stopband Attenuation	f > 0.584fs	-60			dB
ADC Filter Type 1 (USB mode, 272fs or Normal mode operation)					
Passband	+/- 0.05dB	0		0.4535fs	
	-6dB		0.5fs		
Passband Ripple				+/- 0.05	dB
Stopband		0.5465fs			
Stopband Attenuation	f > 0.5465fs	-60			dB
High Pass Filter Corner Frequency	-3dB		3.7		Hz
	-0.5dB		10.4		
	-0.1dB		21.6		
DAC Filter Type 0 (USB mode, 250fs operation)					
Passband	+/- 0.03dB	0		0.416fs	
	-6dB		0.5fs		
Passband Ripple				+/-0.03	dB
Stopband		0.584fs			
Stopband Attenuation	f > 0.584fs	-50			dB
DAC Filter Type 1 (USB mode, 272fs or Normal mode operation)					
Passband	+/- 0.03dB	0		0.4535fs	
	-6dB		0.5fs		
Passband Ripple				+/- 0.03	dB
Stopband		0.5465fs			
Stopband Attenuation	f > 0.5465fs	-50			dB

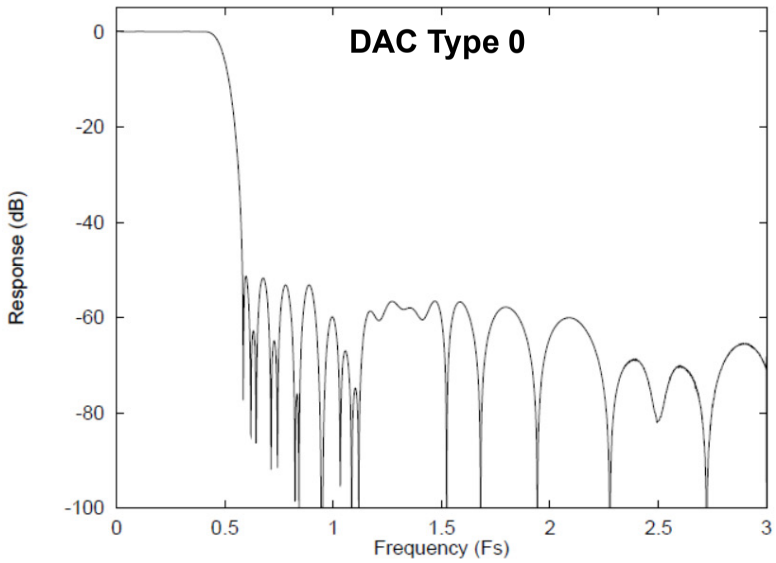
Group Delay Maximums (relative to Fs)

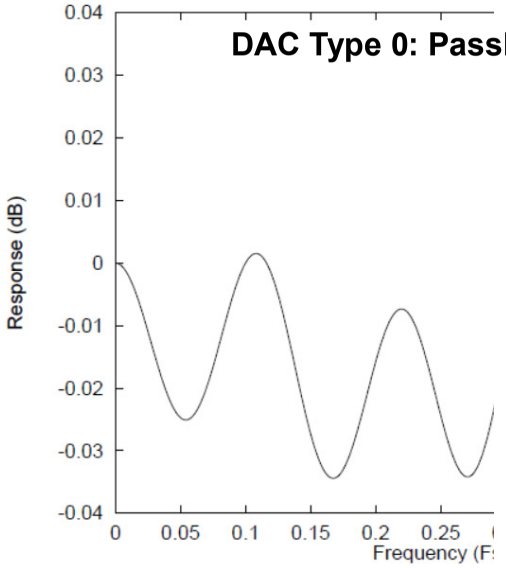
DAC FILTERS		ADC FILTERS	
Mode	Group Delay	Mode	Group Delay
0	11/FS	0	12/FS
1	18/FS	1	20/FS
2	5/FS	2	3/FS
3	5/FS	3	6/FS

- For the WM8731 decimation and reconstruction filter characteristics are also available (test at fs = 48 ksps Type 0 filter)


ADC Type 0
1
1.5
2
2.5
3
Frequency (Fs)


ADC Type 0: Passband Zoom
Response (dB)
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Frequency (Fs)

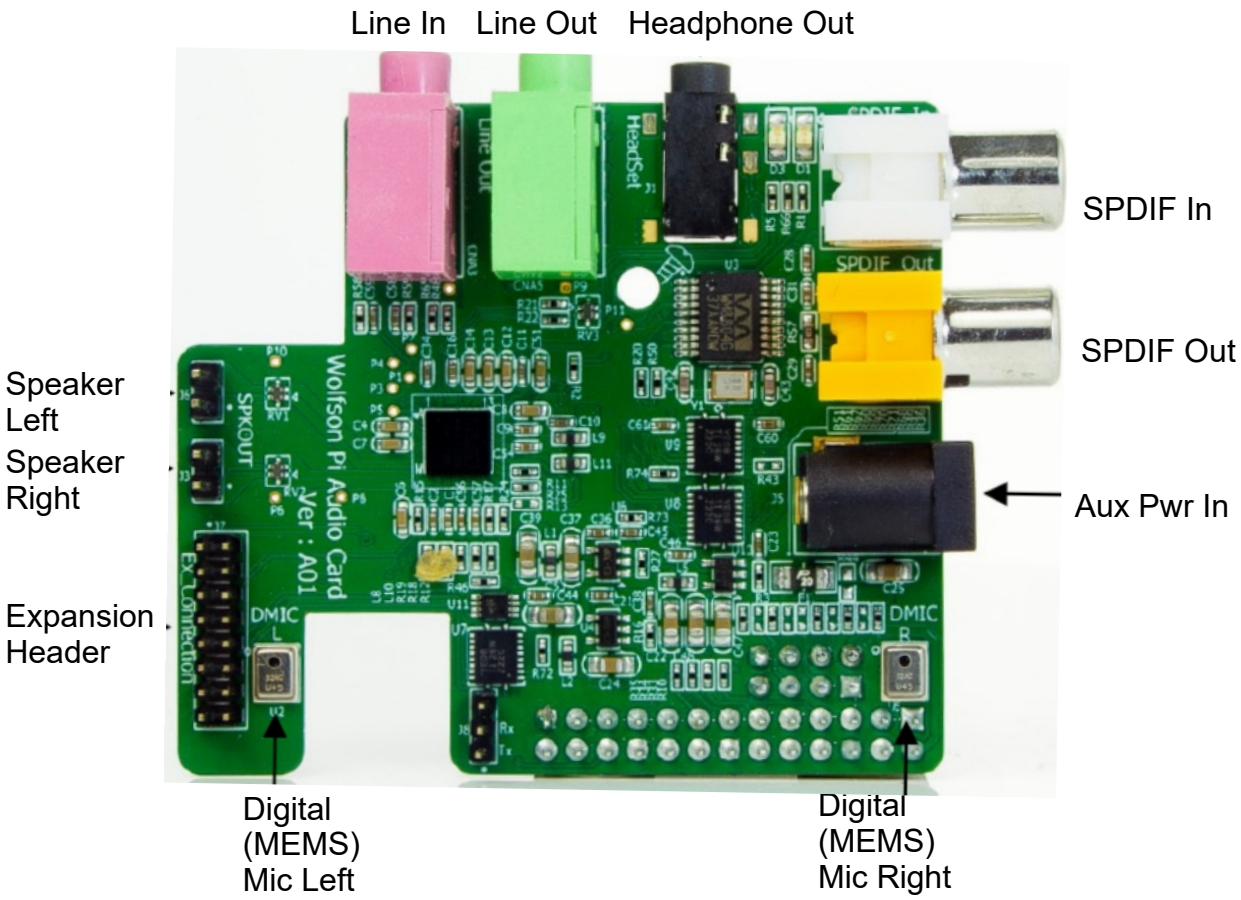

DAC Type 0
Response (dB)
0
-20
-40
-60
-80
-100
0
0.5
1
1.5
2
2.5
3
Frequency (Fs)


DAC Type 0: Passband Zoom
Response (dB)
0.04
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
0
0.05
0.1
0.15
0.2
0.25
Frequency (F

Wolfson 5102 Audio Codec on the STM32F4

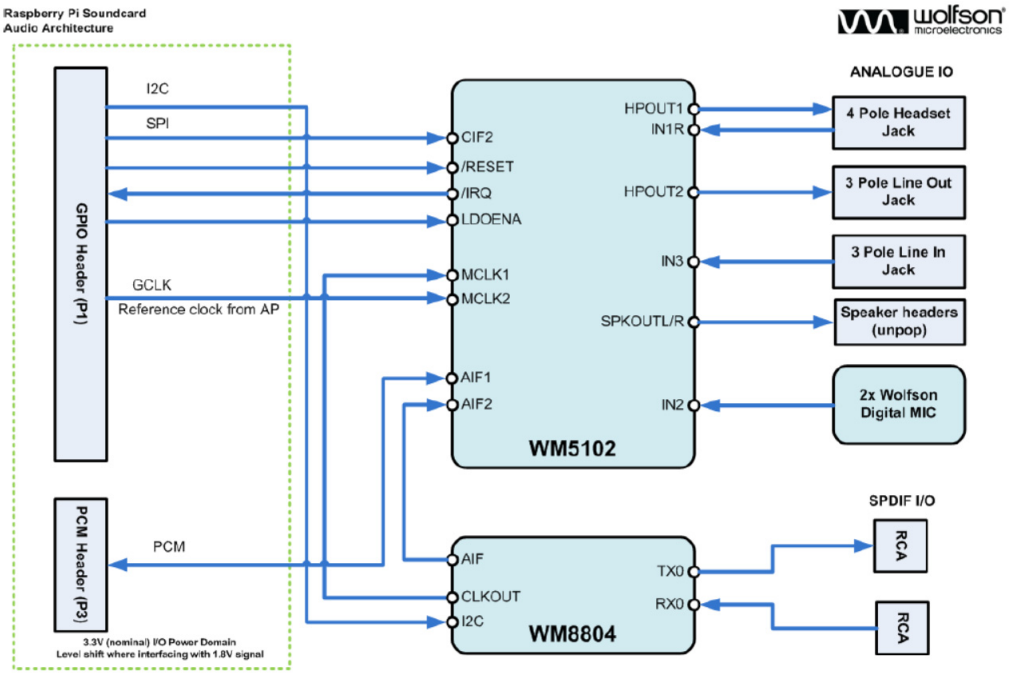
- We now provide some details about the internals of the Pi Audio card with regard to ADC and DAC characteristics

PI Audio Board Layout Details

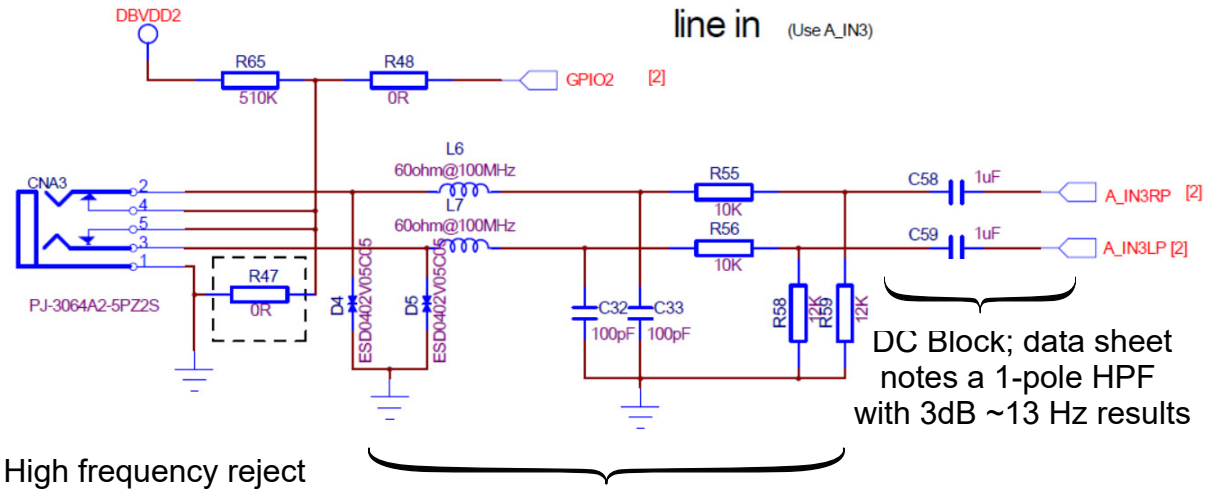


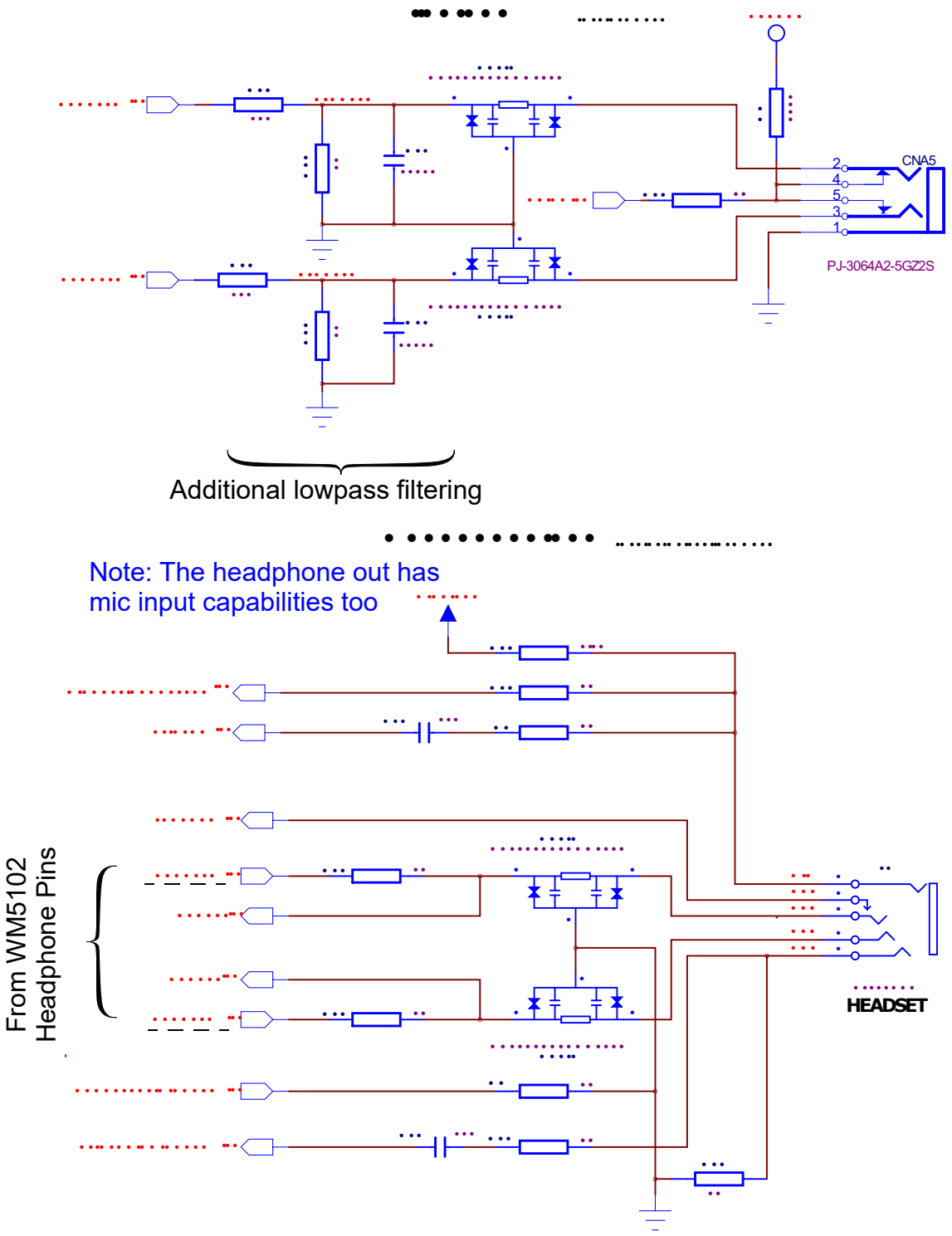
- The ports we use most frequently in this course are:
  - The line input
  - The line output
  - The headphone output
- Projects may find the interface to the digital microphones useful for a simple array processing

- The functional block diagram of the complete Pi Audio card is shown below:



- The second chip, WM8804, is for the Sony/Philips Digital Interface Format (SPDIF) audio interface which is not of concern at present
- The circuit interfaces give some insight into additional filtering in the analog domain




CNA5
2
4
5
3
1
PJ-3064A2-5GZ2S
Additional lowpass filtering
Note: The headphone out has
mic input capabilities too
From WM5102
Headphone Pins
HEADSET

Analog I/O Interface Specs

- Interface performance specs for the line in/out and headphone jack are listed below:

Line Output			
Parameter	**Description/Conditions**	**Typical Value**	**Units**
Connector	Electrical output via 3 pole 3.5mm Socket		
Rout	Output impedance	16	Ohms
Cload	Max capacitive load on output	2	nF
Rload		10	kOhms
Vout	Full scale output signal level	1	Vrms
Vnoise	Noise Floor, no signal applied, A-weighting filter	4.5	uVrms
THD	Total Harmonic Distortion	0.005	%
Notes	Test conditions as follows unless otherwise noted: 48kHz sample rate, test signal 1kHz sine wave, bandwidth measured 20Hz to 20kHz, Rload = 10kOhms		

Line Input			
Parameter	**Description**	**Typical Value**	**Units**
Connector	Electrical input via 3 pole 3.5mm Socket		
Rin	Input Impedance	16	kOhms
Fcut	-3dB down from 1kHz signal level at this frequency	13	Hz
Vin	Full scale input signal	1	Vrms
Vnoise	Equivalent input noise level, no signal applied	20	uVrms
THD	Total Harmonic Distortion @ 0.9Vrms Input	0.035	%
Notes	Test conditions as follows unless otherwise noted: 48kHz sample rate, test signal 1kHz sine wave and bandwidth measured is 20Hz to 20kHz		

Headphone Output			
Parameter	**Description/Conditions**	**Typical Value**	**Units**
Connector	Electrical output via 4 pole 3.5mm Socket		
Rout	Output impedance	0.2	Ohms
Cload	Max capacitive load on output	130	pF
Vout	Full scale output signal level (note that the headphone use case script limits this to 0.5Vrms for safety)	1	Vrms
Vnoise	Noise Floor, no signal applied, A-weighting filter	2.3	uVrms
THD	Total Harmonic Distortion @ 400mVrms output	0.0035	%
Notes	Test conditions as follows unless otherwise noted: 48kHz sample rate, test signal 1kHz sine wave, bandwidth measured 20Hz to 20kHz, Rload = 32Ohms		

Headset Input			
Parameter	**Description**	**Typical Value**	**Units**
Connector	Electrical input via 4 pole 3.5mm Socket, MIC on Sleeve		
Vbias	Microphone voltage bias level	2.8	V
Rbias	Microphone bias resistor	2.2	kOhms
Fcut	-3dB down from 1kHz signal level at this frequency	13	Hz
Vin	Full scale input signal (500mVrms to 14mVrms depending on input gain setting)	63	mVrms
Vnoise	Equivalent input noise level, no signal applied, (settings for 63mVrms signal)	3.2	uVrms
THD	Total Harmonic Distortion @ 57mVrms Input	0.01	%
Notes	Test conditions as follows unless otherwise noted: 48kHz sample rate, test signal 1kHz sine wave and bandwidth measured is 20Hz to 20kHz		

ADC and DAC Frequency Response Specs

- As analog signals are digitized we are interested in the effective antialising filter response of the ADC as well as the effective reconstruction filter frequency response

– Note: Analog filtering elements noted on the schematic, create additional frequency response shaping

Test Conditions

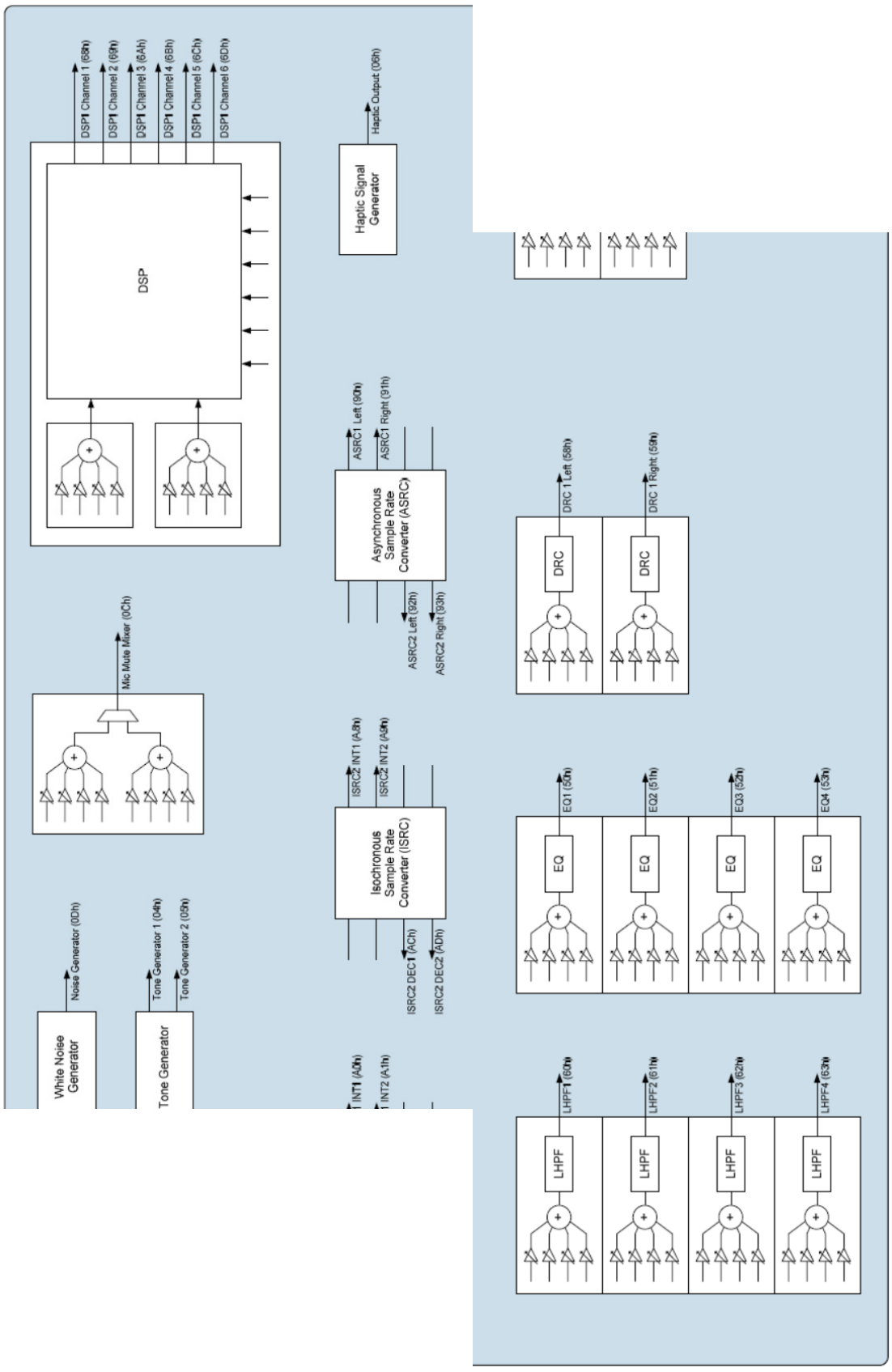
fs ≤ 48kHz

With the exception of the condition(s) noted above, the following electrical characteristics are valid across the full range of recommended operating conditions.

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNIT
ADC Decimation Filters						
Passband		+/- 0.05dB	0		0.454 fs	
		-6dB		0.5 fs		
Passband ripple					+/- 0.05	dB
Stopband			0.546 fs			
Stopband attenuation		f > 0.546 fs	85			dB
Signal path delay		Analogue input to Digital AIF output			2	ms
DAC Interpolation Filters						
Passband		+/- 0.05dB	0		0.454 fs	
		-6dB		0.5 fs		
Passband ripple					+/- 0.05	dB
Stopband			0.546 fs			
Stopband attenuation		f > 0.546 fs	85			dB
Signal path delay		Digital AIF input to Analogue output			1.5	ms

Fixed Function Signal Processing and Programmable DSP

- The WM5102 has much more capability than we make use of in this course
- It contains many fixed signal processing blocks and a programmable DSP
  - For a complete description consult the product data sheet
  - **Note**: Wolfson recently merged with Cirrus Logic
- To give some idea of what is on-board the chip consider the following top-level block diagrams:



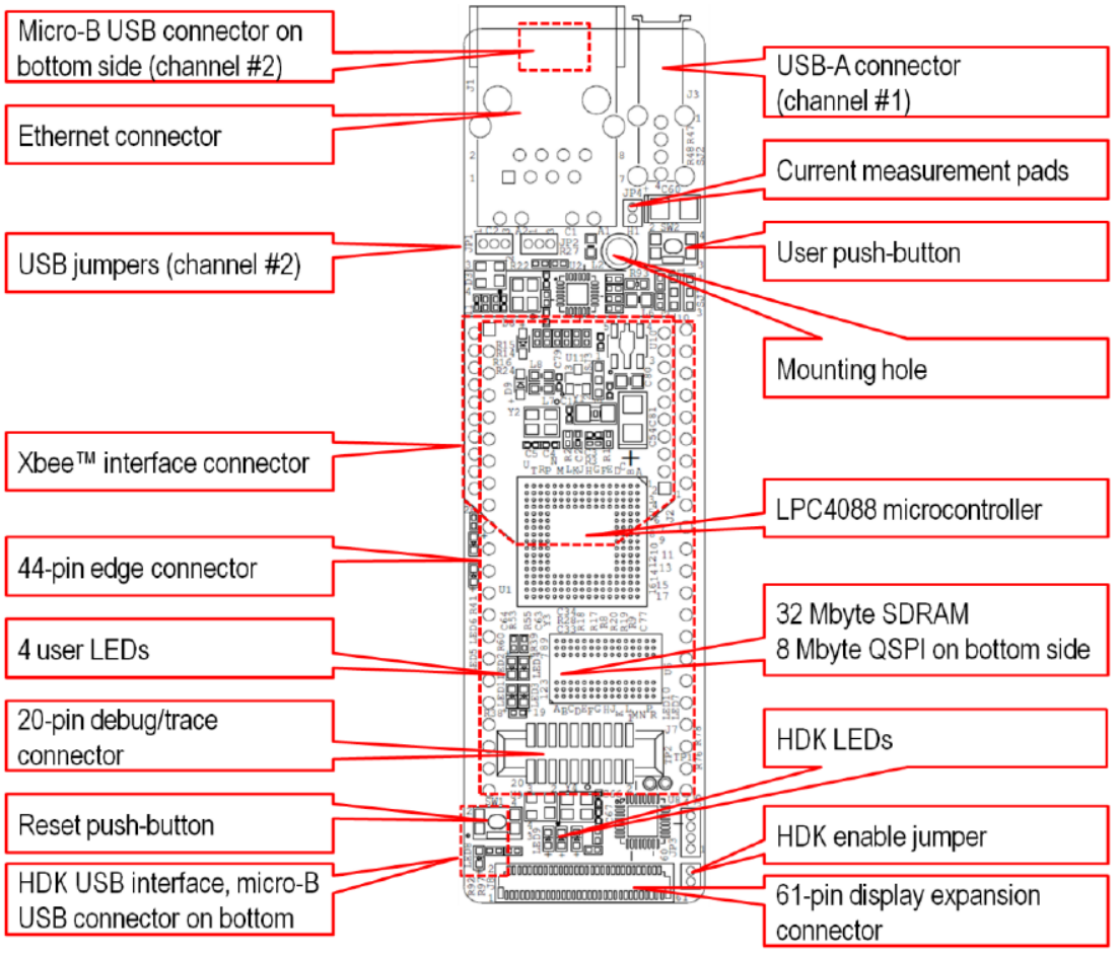
Fixed and Programmable Signal Processing Blocks of the WM5102

Wolfson 8731 Audio Codec on the LPC4088 Baseboard

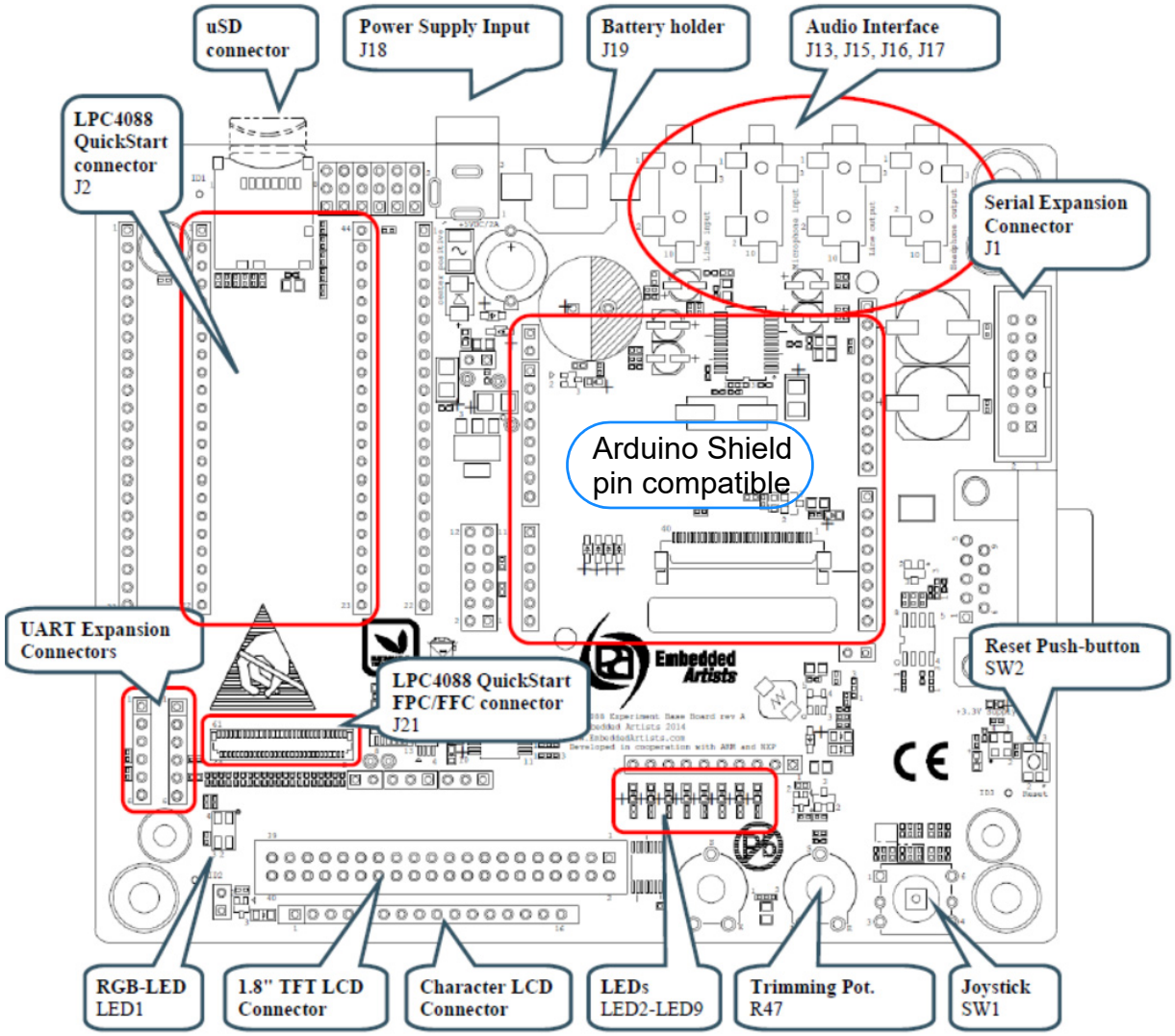
- The Embedded Artists baseboard (EA-QSB-018) for the LPC4088 Quick Start board, offers a wide array of subsystems

LPC4088 Quickstart Interfaces

- To quickly review, the LPC4088 Quickstart is a small gumstick size board that sits over one corner of the base board



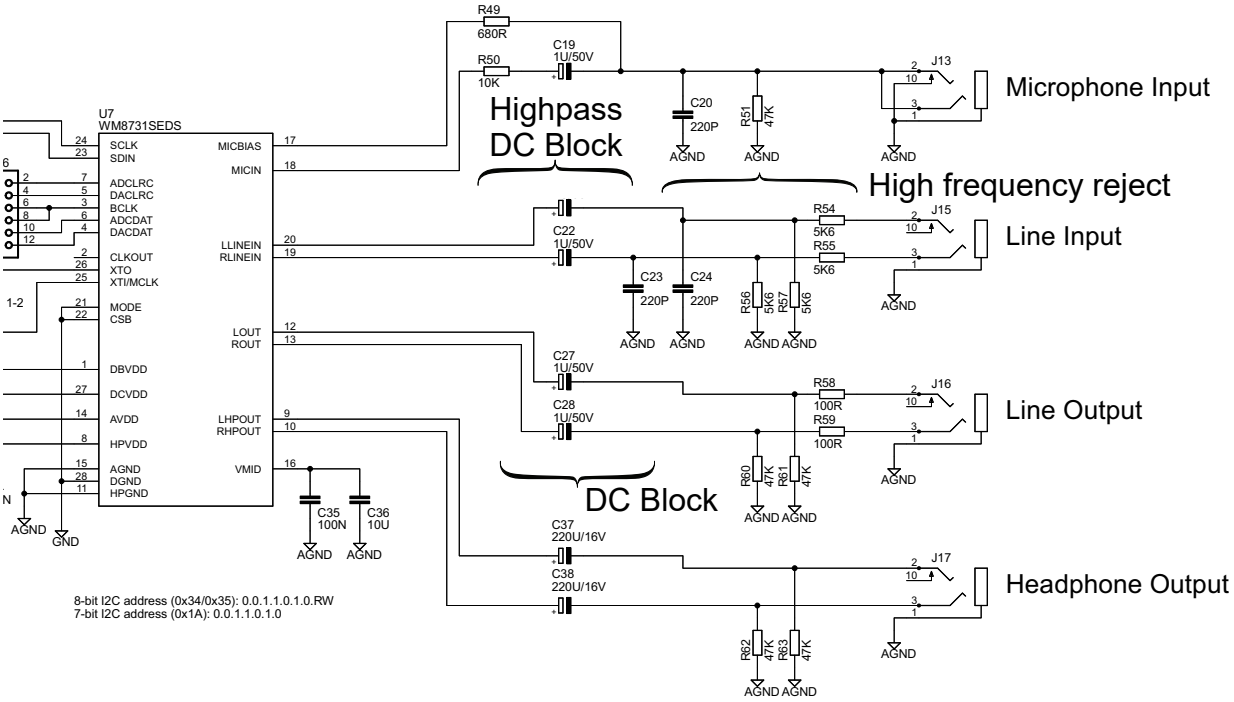
LPC4088 Baseboard Interfaces



Baseboard Schematic

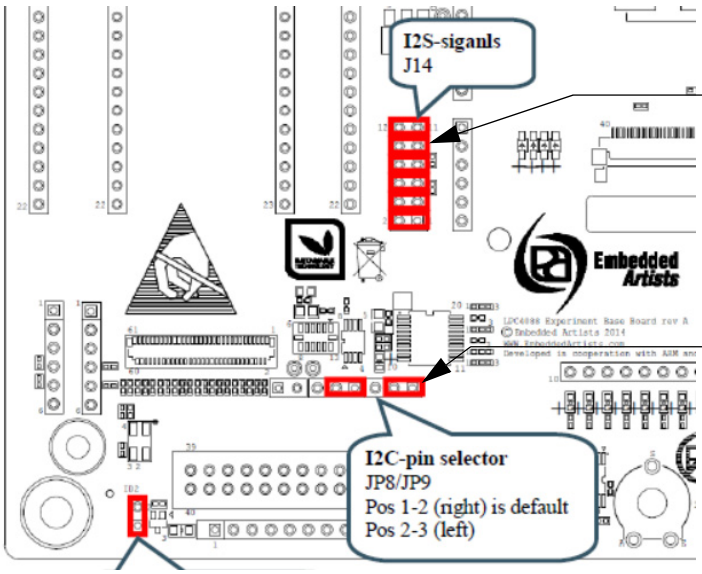
- The LPC4088 baseboard specific circuit interfaces give some

insight into additional filtering in the analog domain



Additonal Baseboard Information

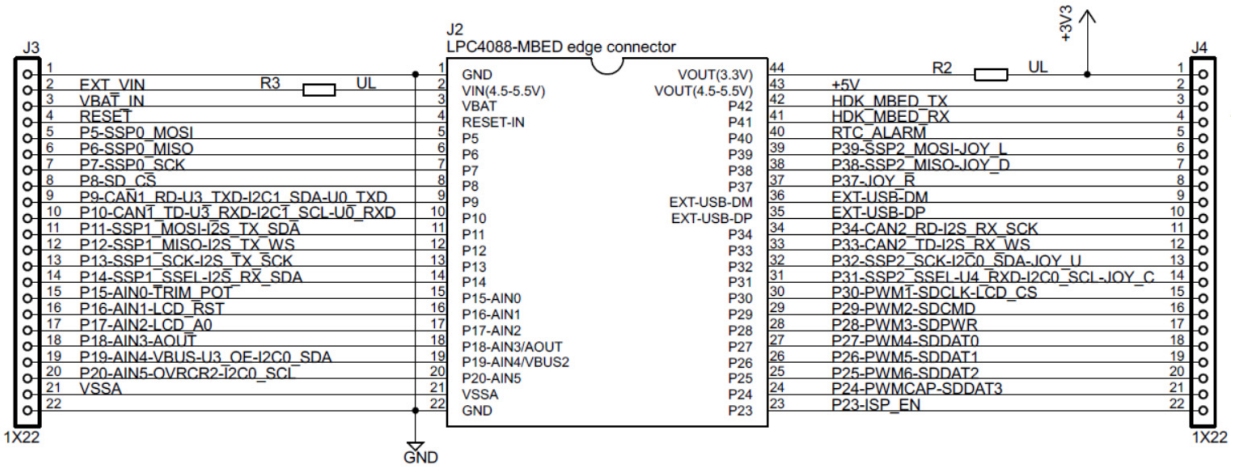
- Default Jumpers and Jumpers Needed for LiB Software



I2S: Leave in default for both LiB and mbed

I2C: Leave in default for mbed, move both jumpers to the alternate settings (left) for LiB codec control

- LPC4088 to Baseboard Pinning



- Additional pinning not shown here, but available, is:
  - Arduino R3 compatible connections
  - LPC4088 to LPC4088 mbed relabeling

Real-Time Processing Options

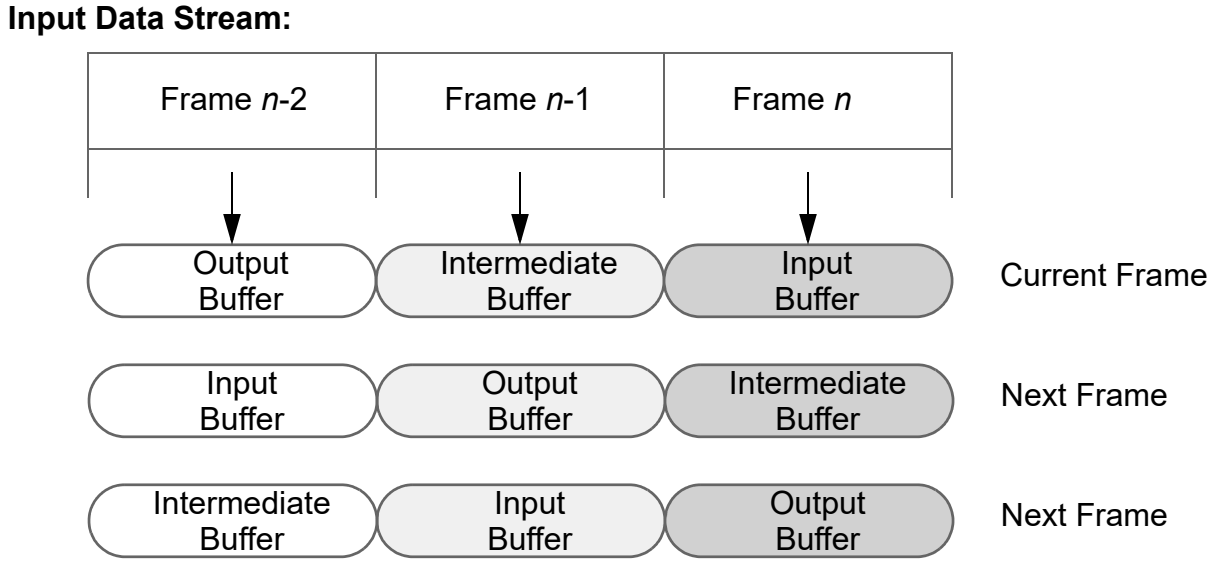
- The most basic of real-time signal processing options is to process a single sample from the ADC though some algorithm and then send it out to the DAC
  - In Welch et.al.[1]this is simply referred to as *sample-based* processing
  - Sample-based processing is easy to follow and write code for
  - If offers minimal latency for the input output sample

1. T.B. Welch, C.H.G. Wright, and M.G. Morrow, *Real-Time Digital Signal Porcessing from MATLAB to C with the TMS320C6x DSPs*, CRC Press, 2012.

stream, but is inefficient as jumping in and out of ISRs to process/filter one sample wastes precious CPU cycles

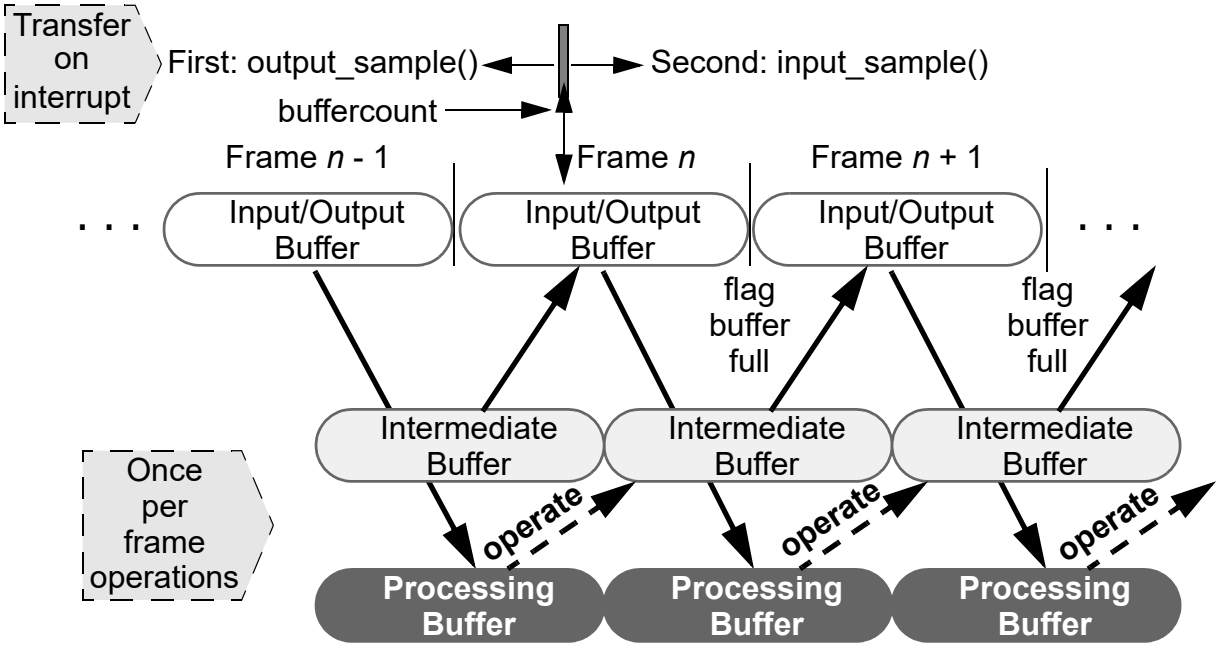
Frame-Based Using ISRs

- The next step up from sample-based approach is to collect samples into frames of samples (*frame-based*), yet still use and ISR handle sample collection to and from the codec
- A downside of any frame-based scheme is that there is inherent latency
  - Think streaming audio over the INTERNET
  - Do you care that a music program (one-way traffic) has some delay?
- Some DSP algorithms, notably the fast Fourier transform (FFT) are frame-based to begin with
- One approach is the triple buffer system shown below:

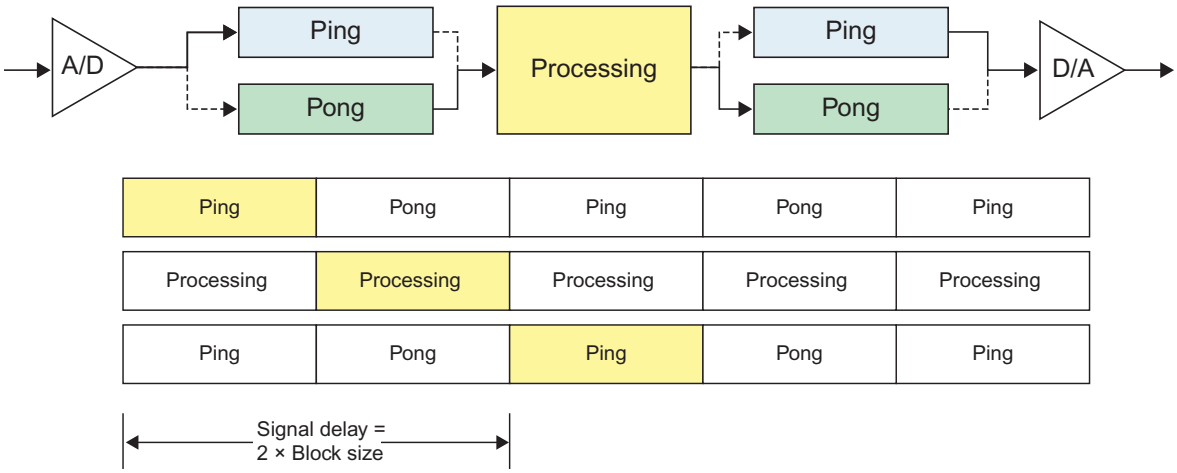


  - The buffers are just blocks of memory

- As the *Input buffer* is filling sample-by-sample from the A/D, the FFT is being computed using the *Intermediate Buffer* as a complete frame of data
- At the same time the output buffer is writing its contents out sample-by-sample to the D/A
- Note that with this scheme there is an inherent two frame delay or lag
- A variation on the above scheme is to use a single input/output buffer, one intermediate buffer, and a primary buffer



- In this scheme each of the three buffers assumes the same role on each frame cycle, while in the previous scheme the roles changed periodically (three frame period)
- Another scheme is to use four buffers, two for input (ping_in/pong_in) and two for output (ping_out/pong_out)[1]



Frame-Based Using Direct Memory Access (DMA)

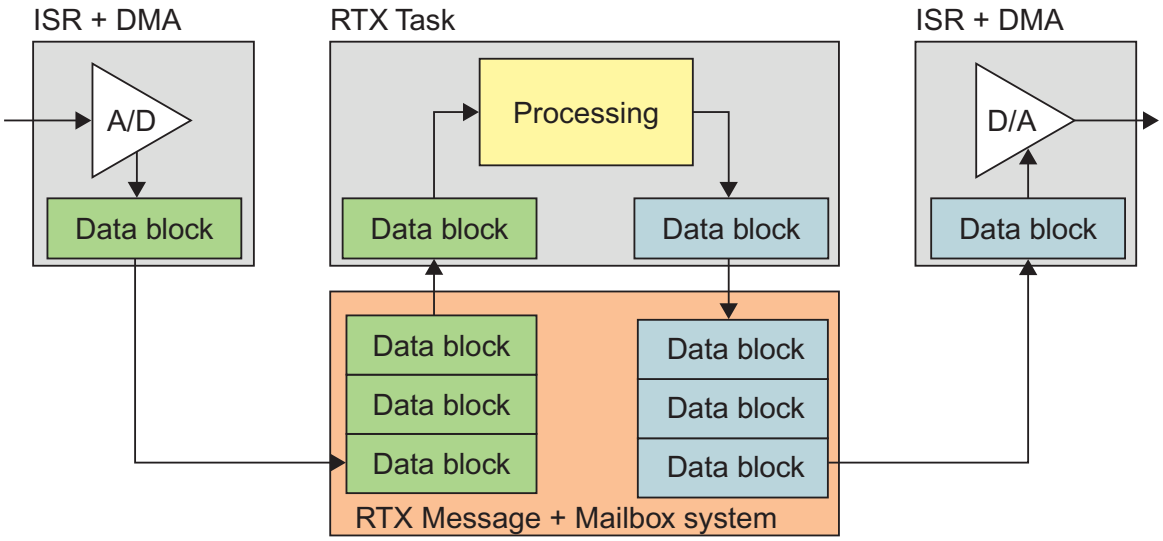
- Dedicated DSP processors as well as the Cortex-M4 include special purpose hardware, namely the *direct memory access controller*, to move blocks of samples to and from memory without burdening the CPU/DSP
- The DMA controller interrupts the CPU only when transfers are complete, thus leaving the CPU free to carry on DSP number crunching
- The LiB software provides an example of frames-based processing using DMA with ping and pong buffers

A RTOS (Real-Time Operating System) Approach

- Beyond the use of DMA, once might consider setting up an RTOS (part of CMSIS) to provided a mail-box system (T.

1. T. Martin, *The Designer's Guide to the Cortex-M Processor Family A Tutorial Approach*, Newnes, 2013.

Martin)

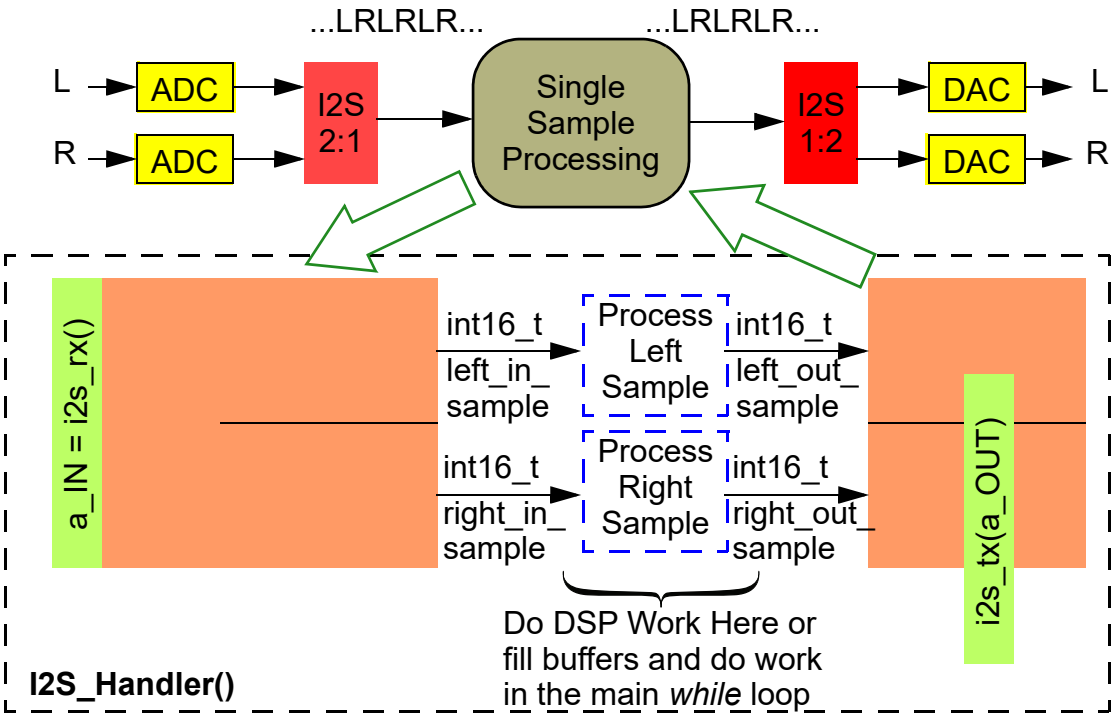


– This approach combines ISR and DMA together as well

– See Martin for a detailed example on this approach

Cypress FM4 Interrupt Program for Sample-Based Processing

- This is the scheme that was introduced the first day of class
- A block diagram of the signal sample flow in the context of the `I2S_HANDLER()` is shown below:




```
audio_chL =
   (audio_IN &
   0x0000FFFF);
```

```
audio_chR =
   ((audio_IN >>16)&
   0x0000FFFF);
```

```
i2s_tx(a_OUT)
```

Unpack 32-bit integer into 3 16-bit

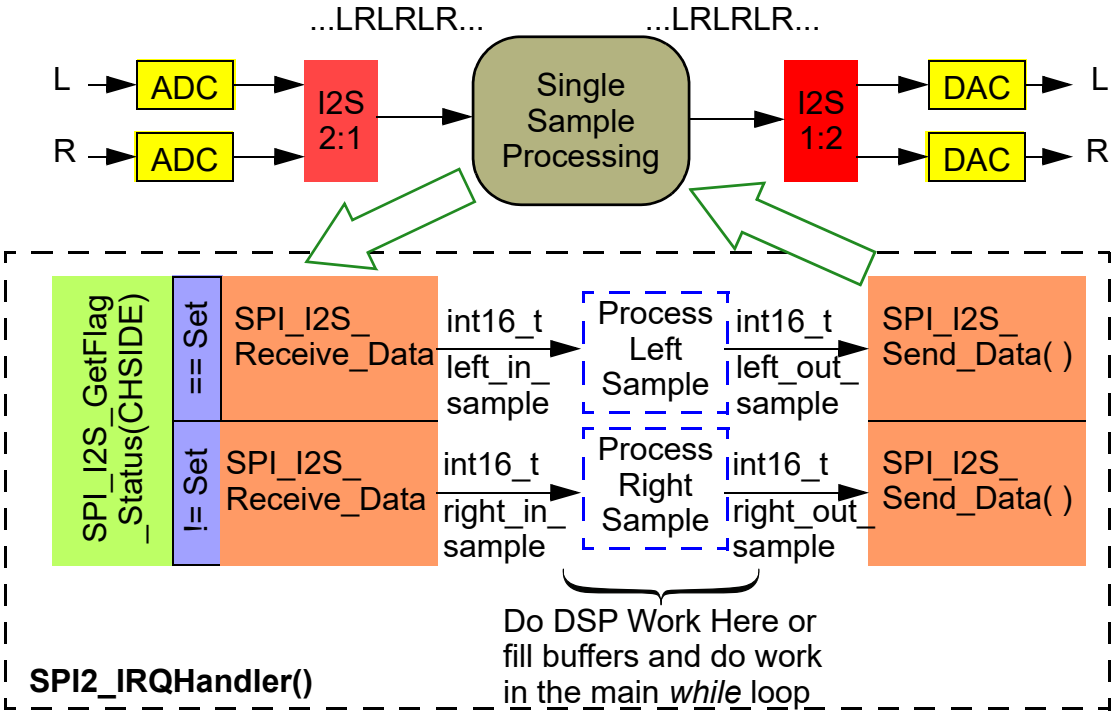
```
a_chR = ((a_IN >>16)& 0x0000FFFF);

a_chL = (a_IN & 0x0000FFFF);


audio_OUT = ((audio_chR<<16 & 0xFFFF0000))
          + (audio_chL & 0x0000FFFF);
```

Drawing Incomplete, work in Progress

- TBD, but similar to the STM32F4.

STM32F4 Interrupt Program for Sample-Based Processing

- This is the scheme that was introduced the first day of class
- A block diagram of the signal sample flow in the context of the software interface is shown below:



- The software takes the form

```
// stm32_Codeclab_intr.c

#include "defines.h"
#include "tm_stm32f4_delay.h"  //needed for USART library
#include "tm_stm32f4_gpio.h"   //include for GPIO library
#include "tm_stm32f4_usart.h"  //USART library to allow serial port control
#include "stm32_wm5102_init.h" //include for LiB codec library

int32_t rand_int32(void);

#define NUMBER_OF_FIELDS 6 //how many parameters to receive from GUI

//Globals for parameter sliders via serial port on USART6
//Initial conditions set as desired (match GUI control settings is best)
float32_t P_vals[NUMBER_OF_FIELDS] = {1.0,1.0,1.0,1.0,1.0,1.0};
int16_t P_idx;
```

```
int16_t H_found = 0;

void SPI2_IRQHandler() //All DSP algorithms in here unless using frame buffers
{
  int16_t left_out_sample = 0;
  int16_t right_out_sample = 0;
  int16_t left_in_sample = 0;
  int16_t right_in_sample = 0;

  TM_GPIO_SetPinHigh(GPIOC, GPIO_Pin_8);

  if (SPI_I2S_GetFlagStatus(I2Sx, I2S_FLAG_CHSIDE) == SET)
  {
    TM_GPIO_SetPinHigh(GPIOC, GPIO_Pin_9);
    left_in_sample = SPI_I2S_ReceiveData(I2Sx);

    //x = (float32_t)(((int16_t)rand_int32())>>1);
    //x = (float32_t) left_in_sample;
    left_out_sample = (int16_t) (P_vals[0]*left_in_sample);

    while (SPI_I2S_GetFlagStatus(I2Sxext, SPI_I2S_FLAG_TXE ) != SET){}
    SPI_I2S_SendData(I2Sxext, left_out_sample);
    TM_GPIO_SetPinLow(GPIOC, GPIO_Pin_9);
  }
  else
  {
    TM_GPIO_SetPinHigh(GPIOC, GPIO_Pin_14);
    right_in_sample = SPI_I2S_ReceiveData(I2Sx);
    right_out_sample = (int16_t)(P_vals[1]*right_in_sample);
    while (SPI_I2S_GetFlagStatus(I2Sxext, SPI_I2S_FLAG_TXE ) != SET){}
    SPI_I2S_SendData(I2Sxext, right_out_sample);
    TM_GPIO_SetPinLow(GPIOC, GPIO_Pin_14);
  }
  TM_GPIO_SetPinLow(GPIOC, GPIO_Pin_8);
}

int main(void)
{
  //Variables for parameter slider communication
  uint8_t c;
  char P_tx[20]; // tx echo chr array
  char P_rcvd[10]; // received chr array
  uint8_t i = 0;
  //Initialize system
  SystemInit();
  /* Initialize delay */
  TM_DELAY_Init();
  //Initialize USART6 at some baud, TX: PC6, RX: PC7
  // Timer error with TM_USART requires a recalc of the baud rate by 231/127
  TM_USART_Init(USART6, TM_USART_PinsPack_1, (231*230400)/127);
  /* Init pins PC8, 9, 14, & 15 for output, no pull-up, and high speed*/
  TM_GPIO_Init(GPIOC, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_14 | GPIO_Pin_15,
               TM_GPIO_Mode_OUT, TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL,
               TM_GPIO_Speed_High);
```

```
  //Finally enable the codec interface with ISR
  stm32_wm5102_init(FS_48000_HZ, WM5102_LINE_IN, IO_METHOD_INTR);
  while(1)
  {
    //Process USART received chars from GUI parameter control app
    //Get character from internal buffer and
    //decode char string into slider float32_t
    //held in P_vals[] array
    c = TM_USART_Getc(USART6);
    if (c) {
      //Wait for header char 'H'
      if (H_found == 0) {
        if (c == 'H') {
          H_found = 1;
        }
      }
      else {
      //TM_USART_Putc(USART6, c);
        if ((c >= '0' && c <= '9') || (c == '.') || (c == '-')) {
          P_rcvd[i] = c;
          i++;
        }
        else if (c == ':') {
          P_idx = (int16_t) atoi(P_rcvd);
          i = 0;
          memset(P_rcvd, 0, sizeof(P_rcvd)); //clear received char array
        }
        else if (c == 'T') {
          P_vals[P_idx] = (float32_t) atof(P_rcvd);
          //For debug echo parameter value back to GUI
          sprintf(P_tx, "H%d:%1.3fT", P_idx,P_vals[P_idx]);
          TM_USART_Puts(USART6, P_tx);
          memset(P_rcvd, 0, sizeof(P_rcvd));
          i = 0;
          H_found = 0;
        }
        else {
          memset(P_rcvd, 0, sizeof(P_rcvd));
        }
      }
    }
  }
}

//Random number generator for white noise testing
int32_t rand_int32(void)
{
  static int32_t a_start = 100001;
  a_start = (a_start*125) % 2796203;
  return a_start;
}
```

- The last line before the main `while()` loop configures the

codec and enables the ISR

- The software behind the codec seen in the main module above is found in `stm32_wm5102_init.c` and `stm32_wm5102_init.h`
- When `stm32_wm5102_init()` is called codec parameters are set the ISR is enabled
- You can set the sampling rate, the input type, and the IO type:
  - The sampling rate options from the `.h` file are

```
#define FS_8000_HZ 0x11
//#define FS_11025_HZ 0x09
#define FS_12000_HZ 0x01
#define FS_16000_HZ 0x12
//#define FS_22050_HZ 0x0A
#define FS_24000_HZ 0x02
#define FS_32000_HZ 0x13
//#define FS_44100_HZ 0x0B
#define FS_48000_HZ 0x03
```

    Note: The commented rates have not been verified!

  - The input types from the `.h` file are

```
#define WM5102_MIC_IN 0
#define WM5102_LINE_IN 1
#define WM5102_DMIC_IN 2
```

    Note: The `MIC_IN` option enters from the headphone jack

  - The IO type is fixed at `IO_METHOD_INTR` since we are doing sample-based processing, however options include

```
#define IO_METHOD_INTR 0
#define IO_METHOD_DMA 1
#define IO_METHOD_POLL 2
```

- One standout characteristic of the SPI2_IRQHandler() as depicted in block diagram above, is that it maintains the character of the I2S audio interface

- int16_t left and right channel samples are brought into the ISR at an update rate that is twice the desired sample rate, e.g., for f_s = 48 kHz the ISR fires at 96 kHz
- This corresponds to the 2:1 and 1:2 I2S muxing block in the block diagram
- Even samples are from the left audio channel and odd samples are from the right audio channel
- Its seems that during debugging operations the channels may flip and then flip back; **be aware of this**

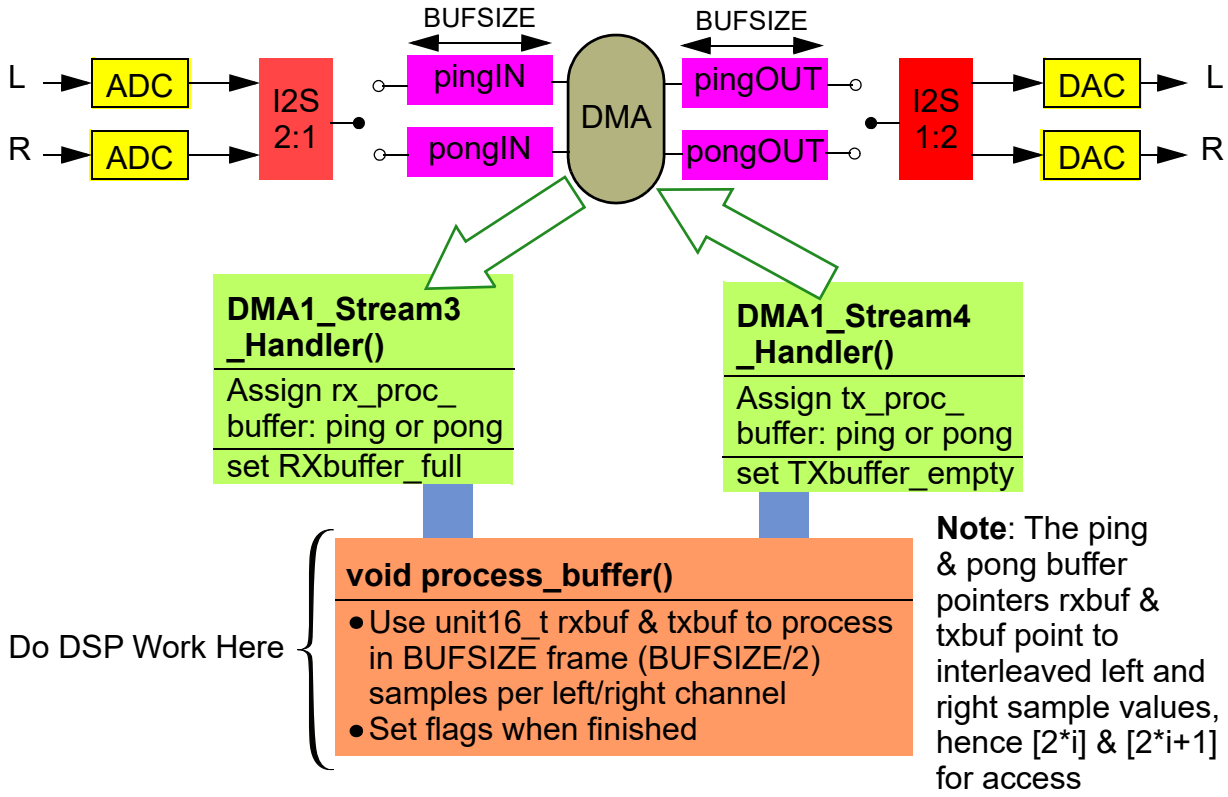
Cypress FM4 DMA Program for Frame-Based Processing

- TBD, but similar to the STM32F4

STM32F4 DMA Program for Frame-Based Processing

- The DMA processing algorithm for the STM32F4 is considerably more complicated because of the use of ping and pong buffers (both input and output) and the DMA framework

- A block diagram of the scheme is shown below:



- The software takes the form:

```
// stm32f4_Codeclab_dma.c

#include "defines.h"
#include "tm_stm32f4_delay.h"  //needed for USART library
#include "tm_stm32f4_gpio.h"   //include for GPIO library
#include "tm_stm32f4_usart.h"  //USART library to allow serial port control
#include "stm32_wm5102_init.h" //include for LiB codec library

int32_t rand_int32(void);

#define NUMBER_OF_FIELDS 6 //how many parameters to receive from GUI

//Globals for parameter sliders via serial port on USART6
//Initial conditions set as desired (match GUI control settings is best)
float32_t P_vals[NUMBER_OF_FIELDS] = {1.0,1.0,1.0,1.0,1.0,1.0};
int16_t P_idx;
int16_t H_found = 0;

//DMA buffers and flags
extern int16_t pingIN[BUFSIZE], pingOUT[BUFSIZE],
               pongIN[BUFSIZE], pongOUT[BUFSIZE];
```

```
int16_t rx_proc_buffer, tx_proc_buffer;
volatile int16_t RX_buffer_full = 0;
volatile int16_t TX_buffer_empty = 0;

float32_t x[BUFSIZE/2], y[BUFSIZE/2]; //working buffers for input and output

void DMA1_Stream3_IRQHandler()
{
  if(DMA_GetITStatus(DMA1_Stream3,DMA_IT_TCIF3))
  {
    DMA_ClearITPendingBit(DMA1_Stream3,DMA_IT_TCIF3);
    if(DMA_GetCurrentMemoryTarget(DMA1_Stream3))
    {
      rx_proc_buffer = PING;
    }
    else
    {
      rx_proc_buffer = PONG;
    }
    RX_buffer_full = 1;
  }
}

void DMA1_Stream4_IRQHandler()
{
  if(DMA_GetITStatus(DMA1_Stream4,DMA_IT_TCIF4))
    {
      DMA_ClearITPendingBit(DMA1_Stream4,DMA_IT_TCIF4);

      if(DMA_GetCurrentMemoryTarget(DMA1_Stream4))
      {
        tx_proc_buffer = PING;
      }
      else
      {
        tx_proc_buffer = PONG;
      }
      TX_buffer_empty = 1;
   }
}

void process_buffer() //All DSP algorithms in here
{
  int i;
  int16_t *rxbuf, *txbuf;

  if (rx_proc_buffer == PING) rxbuf = pingIN; else rxbuf = pongIN;
    if (tx_proc_buffer == PING) txbuf = pingOUT; else txbuf = pongOUT;

  //Processing of the frame samples
```

```
  //Here we are simpling appling a gain scale on each channel from P_vals[]
  for (i=0 ; i<(BUFSIZE/2) ; i++)
  {
    txbuf[2*i] = (int16_t)(P_vals[1] * rxbuf[2*i]);
    //txbuf[2*i] = (int16_t) y1;
    txbuf[2*i+1] = (int16_t)(P_vals[0] * rxbuf[2*i+1]);
    //txbuf[2*i+1] = rxbuf[2*i+1];
  }
  TX_buffer_empty = 0;
  RX_buffer_full= 0;
}

int main(void)
{
  //Variables for parameter slider communication
  uint8_t c;
  char P_tx[20]; // tx echo chr array
  char P_rcvd[10]; // received chr array
  uint8_t i = 0;
  //Initialize system
  SystemInit();
  /* Initialize delay */
  TM_DELAY_Init();
  //Initialize USART6 at some baud, TX: PC6, RX: PC7
  // Timer error with TM_USART requires a recalc of the baud rate by 231/127
  TM_USART_Init(USART6, TM_USART_PinsPack_1, (231*230400)/127);
  /* Init pins PC8, 9, 14, & 15 for output, no pull-up, and high speed*/
  TM_GPIO_Init(GPIOC, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_14 | GPIO_Pin_15,
               TM_GPIO_Mode_OUT, TM_GPIO_OType_PP, TM_GPIO_PuPd_NOPULL,
               TM_GPIO_Speed_High);
  stm32_wm5102_init(FS_48000_HZ, WM5102_LINE_IN, IO_METHOD_DMA);
  while(1)
  {
    while(!(RX_buffer_full && TX_buffer_empty))
    {
      //Get character from internal buffer and
      //decode char string into slider float32_t
      //held in P_vals[] array
      c = TM_USART_Getc(USART6);
      if (c) {
        //Wait for header char 'H'
          if (H_found == 0) {
            if (c == 'H') {
              H_found = 1;
            }
          }
          else {
            //TM_USART_Putc(USART6, c);
            if ((c >= '0' && c <= '9') || (c == '.') || (c == '-')) {
            P_rcvd[i] = c;
```

```
            i++;
          }
          else if (c == ':') {
            P_idx = (int16_t) atoi(P_rcvd);
            i = 0;
            memset(P_rcvd, 0, sizeof(P_rcvd)); //clear received char array
          }
          else if (c == 'T') {
            P_vals[P_idx] = (float32_t) atof(P_rcvd);
            //For debug echo parameter value back to GUI
            sprintf(P_tx, "H%d:%1.3fT", P_idx,P_vals[P_idx]);
            TM_USART_Puts(USART6, P_tx);
            memset(P_rcvd, 0, sizeof(P_rcvd));
            i = 0;
            H_found = 0;
          }
          else {
            memset(P_rcvd, 0, sizeof(P_rcvd));
          }
        }
      }
    }
    //GPIO_SetBits(GPIOD, GPIO_Pin_15);
    TM_GPIO_SetPinHigh(GPIOC, GPIO_Pin_8);
    process_buffer(); //Do all DSP processing with this call
    TM_GPIO_SetPinLow(GPIOC, GPIO_Pin_8);
    //GPIO_ResetBits(GPIOD, GPIO_Pin_15);
  }
}

//Random number generator for white noise testing
int32_t rand_int32(void)
{
  static int32_t a_start = 100001;
  a_start = (a_start*125) % 2796203;
  return a_start;
}
```

- Outside of setting `IO_METHOD_DMA` in the call to `stm32_wm5102_init()`, all of the signal processing work takes place in the function `process_buffer()`
- The full length of the ping and pong buffers, four total, is `BUFFSIZE`
  - This value is defined in `stm32_wm5102_init.h`:

```
#define BUFSIZE 256
```

Note: This is the total frame length for the interleaved left and right channels; `BUFFSIZE/2` is the frame length for the respective left and right channels that you process

Testing the WM5102

- A combination of lab hardware and software can be used to take measurements on the performance of the STM32F4
  - With the WM5012 configured to loop signals through we can characterize the ADC/DAC path
  - If we internally generate a white noise sequence, we can characterize just the DAC output path
- In Assignment #3 these options are explored in detail

Noise Capture to Characterize DAC

- A simple measurement of the DAC path can be obtained by generating white noise in software and sending it directly to the output, i.e.,

```
x = (float32_t)(((short)rand_int32())>>2);
//x = (float32_t) left_in_sample;
//left_out_sample = (int16_t) (P_vals[0]*left_in_sample);
```

  we can set-up the source with a scale factor of 4 (>>2)
- We can then use GoldWave™ or a similar audio capture program, e.g., Audacity
- Record in stereo if you want to capture two channels
- If possible use a sampling rate that exceeds the DAC sam-

pling rate so the measurement bandwidth is greater than the signal being characterized

ISR Timing Using GPIO

- Both the sample-based and DMA frame-based routimes described about have four GPIO pins enabled for digital output
- You can learn a lot about the characteristics and performance of you DSP code by setting GPIO pins in various ways
- More details will be added

Useful Resources

- TBD