2-Channel DJ Mixer

PROBLEM STATEMENT

Build a **two-channel DJ Mixer** to manipulate the **frequency** and **gain** of audio files passed in from an AUX cord based on **user inputs**.

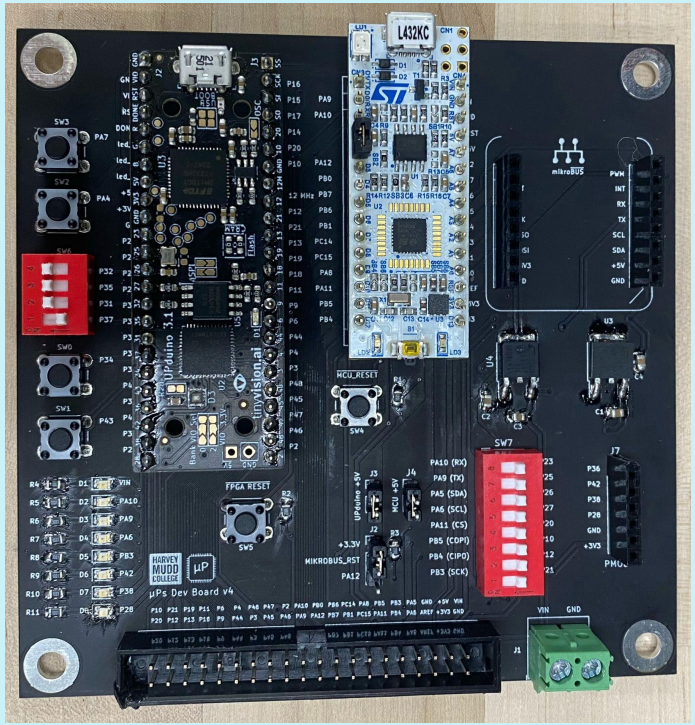
Overview

OBJECTIVES

- Use external ADC to read in audio file
- Communicate audio data from FPGA to MCU
- Implement low and high pass digital filters
- Vary audio gain based on user inputs
- Output manipulated audio onto speaker

Commercial 2-Channel Mixer

Division of Labor



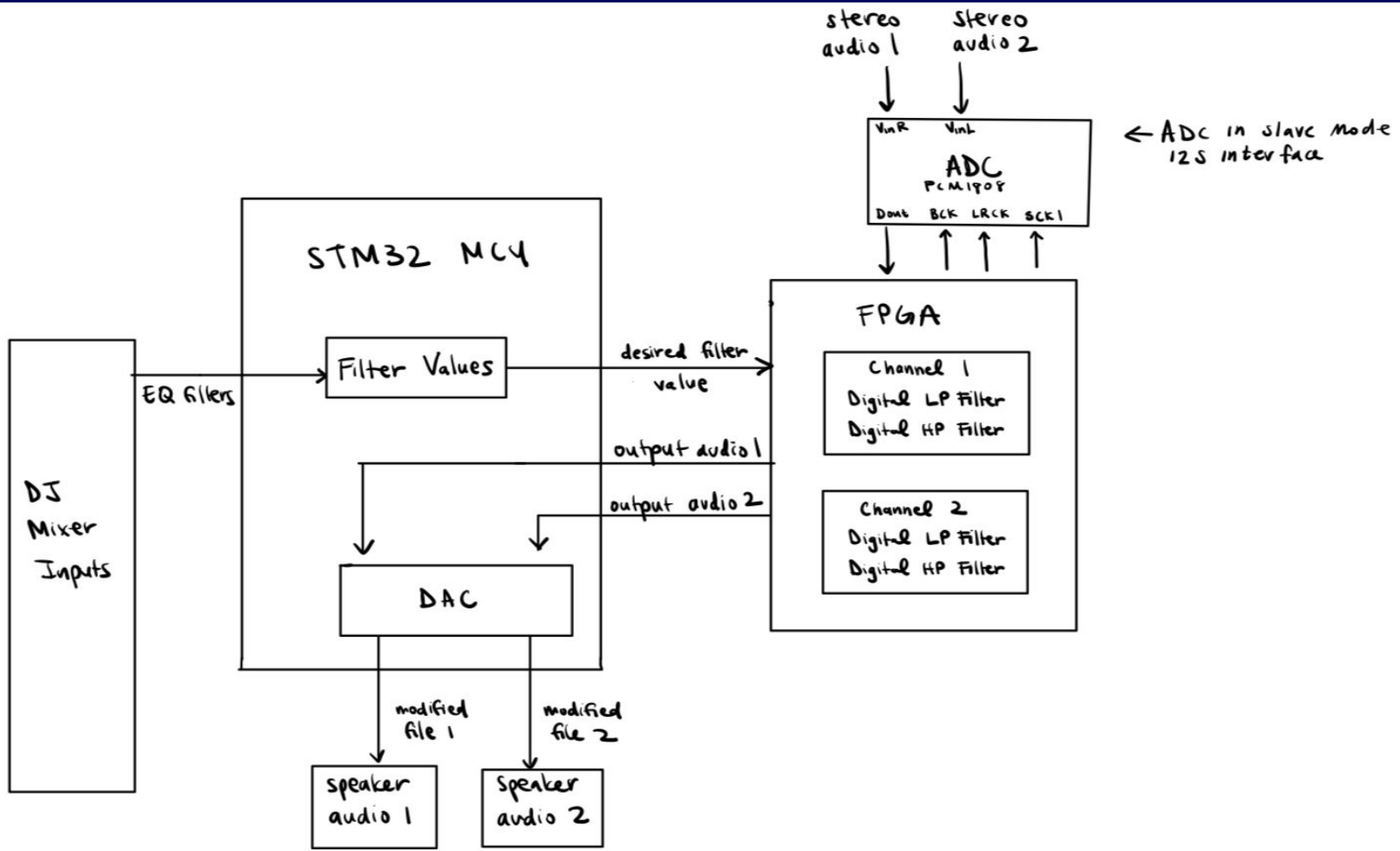
MCU & FPGA on Development Board

STM32L4 MCU

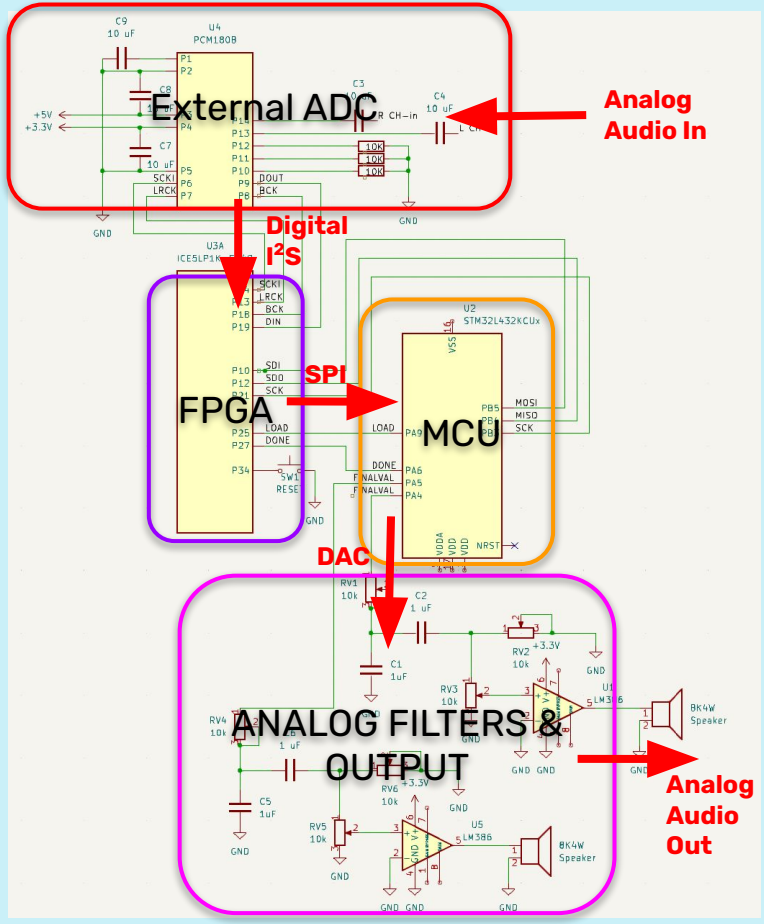
- **Output manipulated audio data (DMA & onboard DAC)**
- **Interpret and transfer user input EQ values to FPGA (onboard ADC & SPI)**
- **Controller for SPI communication**

iCE40 FPGA

- **Generating Clock for PCM1808 ADC to read in audio**
- **Digital filtering of audio data based on EQ values from the MCU**
- **Sending audio to MCU**

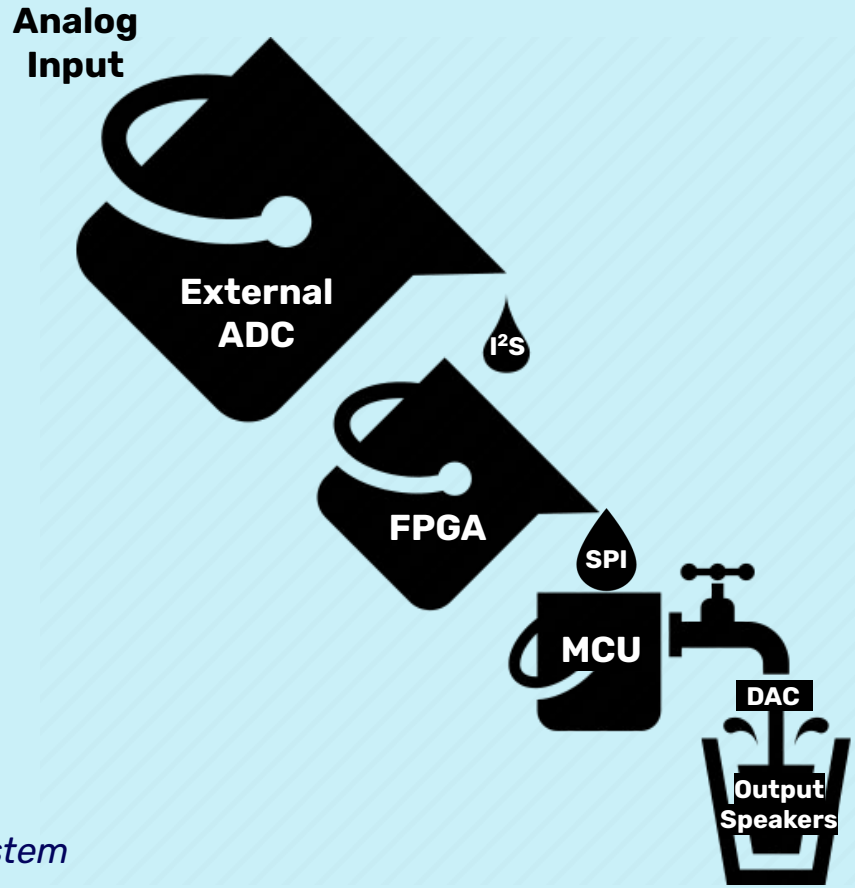

stereo audio 1
stereo audio 2
VinR
VinL
ADC
PCM1808
Dout
BCK
LRCK
SCK1
← ADC in slave mode
I2S interface
STM32 MC4
FPGA
Filter Values
EQ filters
desired filter value
Channel 1
Digital LP Filter
Digital HP Filter
output audio 1
output audio 2
Channel 2
Digital LP Filter
Digital HP Filter
DJ Mixer Inputs
DAC
modified file 1
modified file 2
speaker audio 1
speaker audio 2

Audio Flow



Audio Flow of System on Circuit Diagram

Audio Flow



Audio Flow of System

Sampling Frequency - Nyquist

Sampling Rate

Signal Bandwidth (Max. freq included)

$$f_s > 2B$$

Nyquist Theorem

Human hearing ranges from **20 Hz to 20 kHz** → need to sample at frequency > 40kHz

External ADC

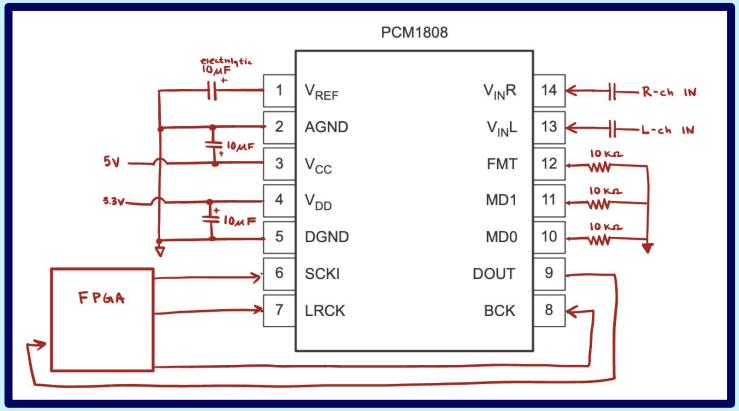
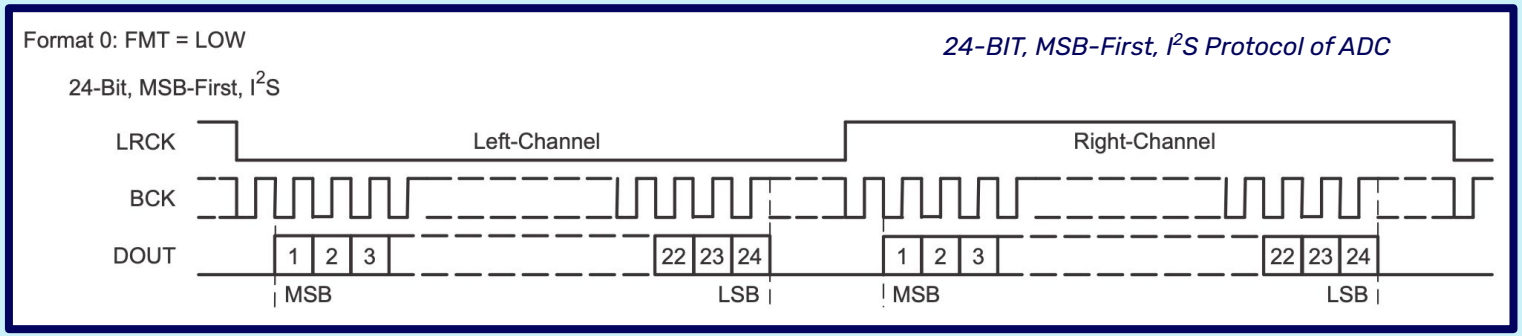


Diagram of PCM1808 ADC and FPGA

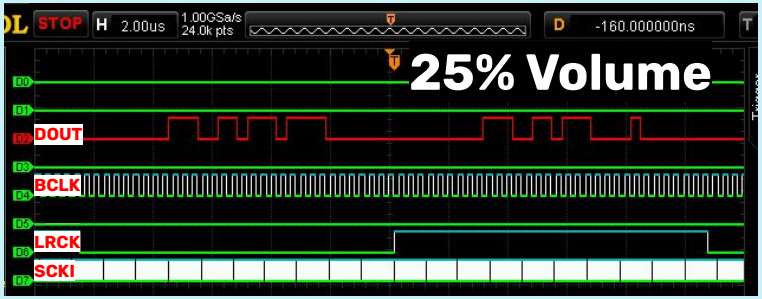
*SCKI = 256 * Fs = 24 MHz*
*BCK = 64 * Fs = 6 MHz*
LRCK = Fs = 93.75 kHz

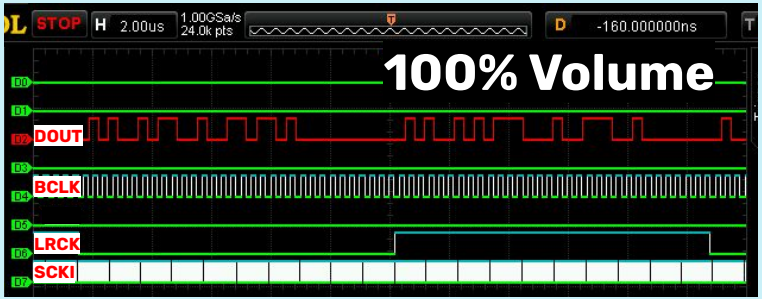
Can sample frequencies up to 46.875 kHz

Clock Inputs into the ADC



I²S Protocol



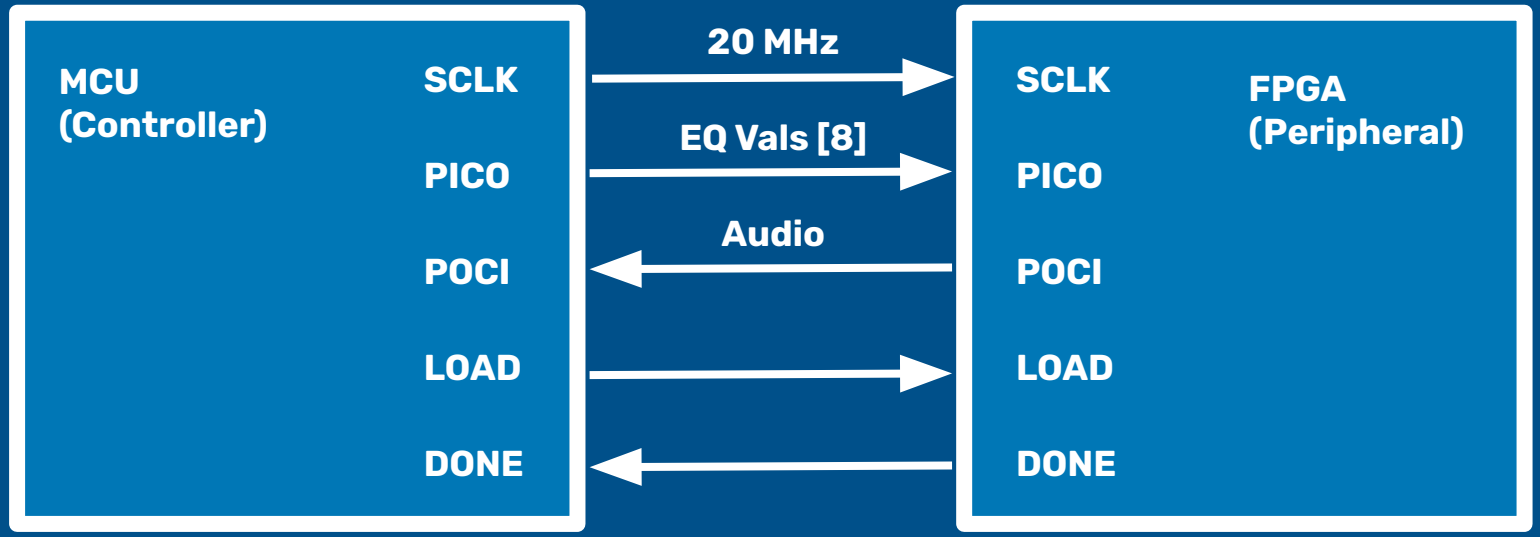


I²S traces ranging from 0% to 100% volume to confirm expected operation from MSB placement

Note: red line is the data out where at higher volume amplitude is larger

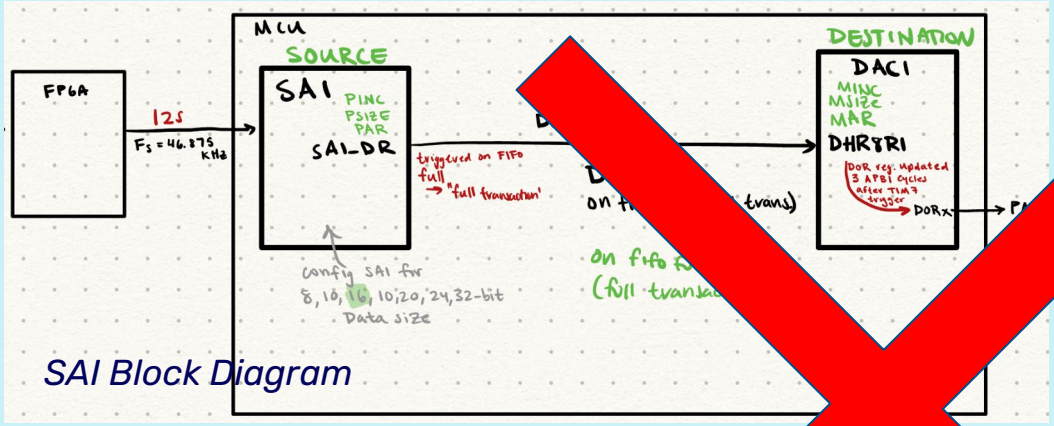
FPGA ↔ MCU SPI

Serial Peripheral Interface

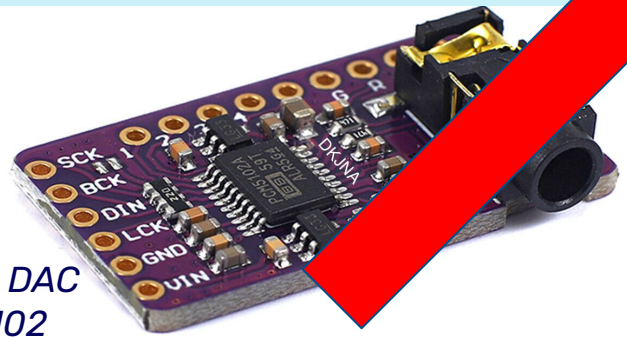


SPI interaction between FPGA & MCU

Why SPI for Audio?



SAI Block Diagram



External DAC
PCM5102

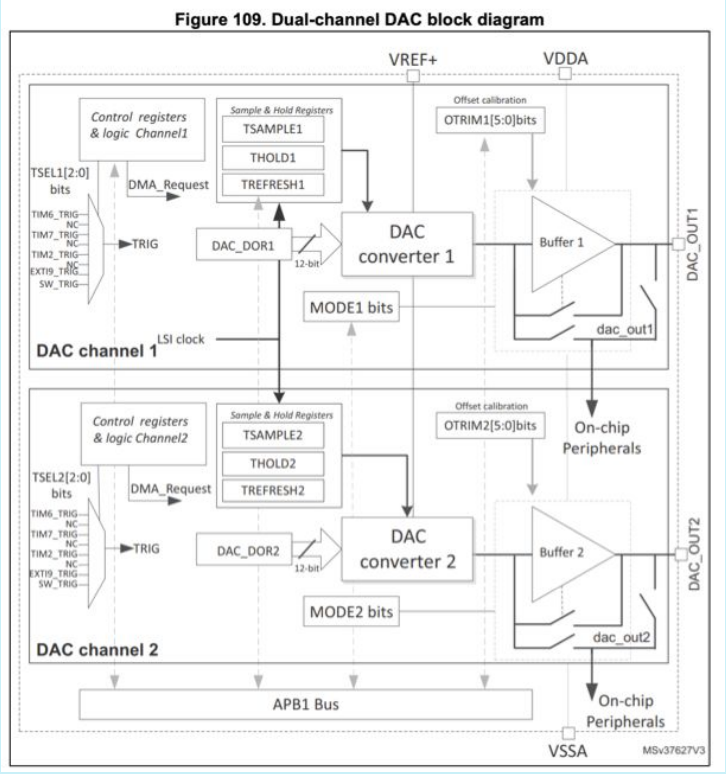
Table 221. SAI register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000		Reserved																															
	SAI_xCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MCKDIV[3:0]				NODIV	Res.	DMAEN	SAIEN	Res.	Res.	OUTDRIV	MONO	SYNCEN[1:0]		CKSTR	LSBFIRST	Res.	DS[2:0]			PRTCFG[1:0]		MODE[1:0]	
	Reset value									0	0	0	0	0		0	0			0	0	0	0	0	0		1	0		0	0	0	0
0x0008 or 0x0028	SAI_xCR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	COMP[1:0]		CPL	MUTECN[5:0]						MUTE VAL	MUTE	TRIS	FFLUS	FTH[2:0]		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x000C or 0x002C	SAI_xFRCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FSOFF	FSPOL	FSDEF		FSALL[6:0]								FRL[7:0]						
	Reset value														0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0x0010 or 0x0030	SAI_xSLOTR	SLOTEN[15:0]																				NBSLOT[3:0]				SLOTSZ[1:0]			FBOFF[4:0]			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0	0	0		0	0	0	0	0
0x0014 or 0x0034	SAI_xIM	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LFSDETIE	AFSDETIE	CNRDYIE	FREQIE	WCKCFGIE	MUTEDETIE	OVRUDRIE
	Reset value																										0	0	0	0	0	0	0
0x0018 or 0x0038	SAI_xSR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FLVL[2:0]			Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LFSDET	AFSDET	CNRDY	FREQ	WCKCFG	MUTEDET	OVRUDR
	Reset value														0	0	0										0	0	0	0	0	0	0
	xCLRFR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CLFSDET	CAFSDET	CCNRDY		CWCKCFG	CMUTEDET	COVRUDR
																											0	0	0		0	0	0
0x0020 or 0x0040		DATA[31:0]																															
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SAI Register Map

MCU: DAC

Block Diagram & DAC Register Map



MCU Dual-Channel DAC Block Diagram

Table 80. DAC register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	DAC_CR	Res.	CEN2	DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[2:0]		TSEL22	TSEL21	TSEL20	TEN2	Res.	EN2	Res.	CEN1	DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[2:0]		TSEL12	TSEL11	TSEL10	TEN1	Res.	EN1
	Reset value		0	0	0	0	0	0	0	0	0	0	0	0	0		0		0	0	0	0	0	0	0	0	0	0	0	0		0
0x04	DAC_SWTRGR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SWTRIG2	SWTRIG1
	Reset value																															0	0
0x08	DAC_DHR12R1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC1DHR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x0C	DAC_DHR12L1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC1DHR[11:0]												Res.	Res.	Res.	Res.
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0				
0x10	DAC_DHR8R1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC1DHR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	DAC_DHR12R2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC2DHR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	DAC_DHR12L2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC2DHR[11:0]												Res.	Res.	Res.	Res.
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0				
0x1C	DAC_DHR8R2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC2DHR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x20	DAC_DHR12RD	Res.	Res.	Res.	Res.	DACC2DHR[11:0]												Res.	Res.	Res.	Res.	DACC1DHR[11:0]											
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0
0x24	DAC_DHR12LD	DACC2DHR[11:0]												Res.	Res.	Res.	Res.	DACC1DHR[11:0]												Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0				
0x28	DAC_DHR8RD	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC2DHR[7:0]								DACC1DHR[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	DAC_DOR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC1DOR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x30	DAC_DOR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DACC2DOR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0

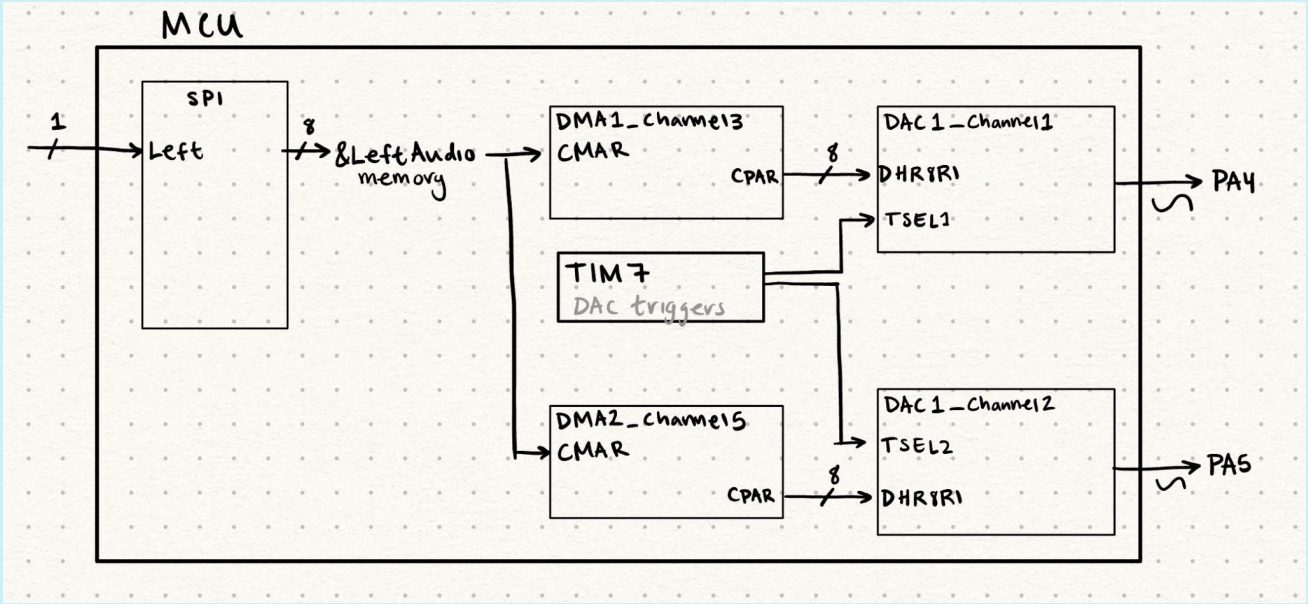
Want audio samples in this register for outputting

MCU DAC Register Map

MCU: DMA & DAC

```
DMA1_Channel3->CPAR = (uint32_t) &(DAC1->DHR8R1); //Place DMA into DAC Outplacement
```

Use DMA to output audio into DAC output register



MCU Block Diagram for Left channel Audio

$$Timer freq = \frac{clk freq}{PSC+1}$$

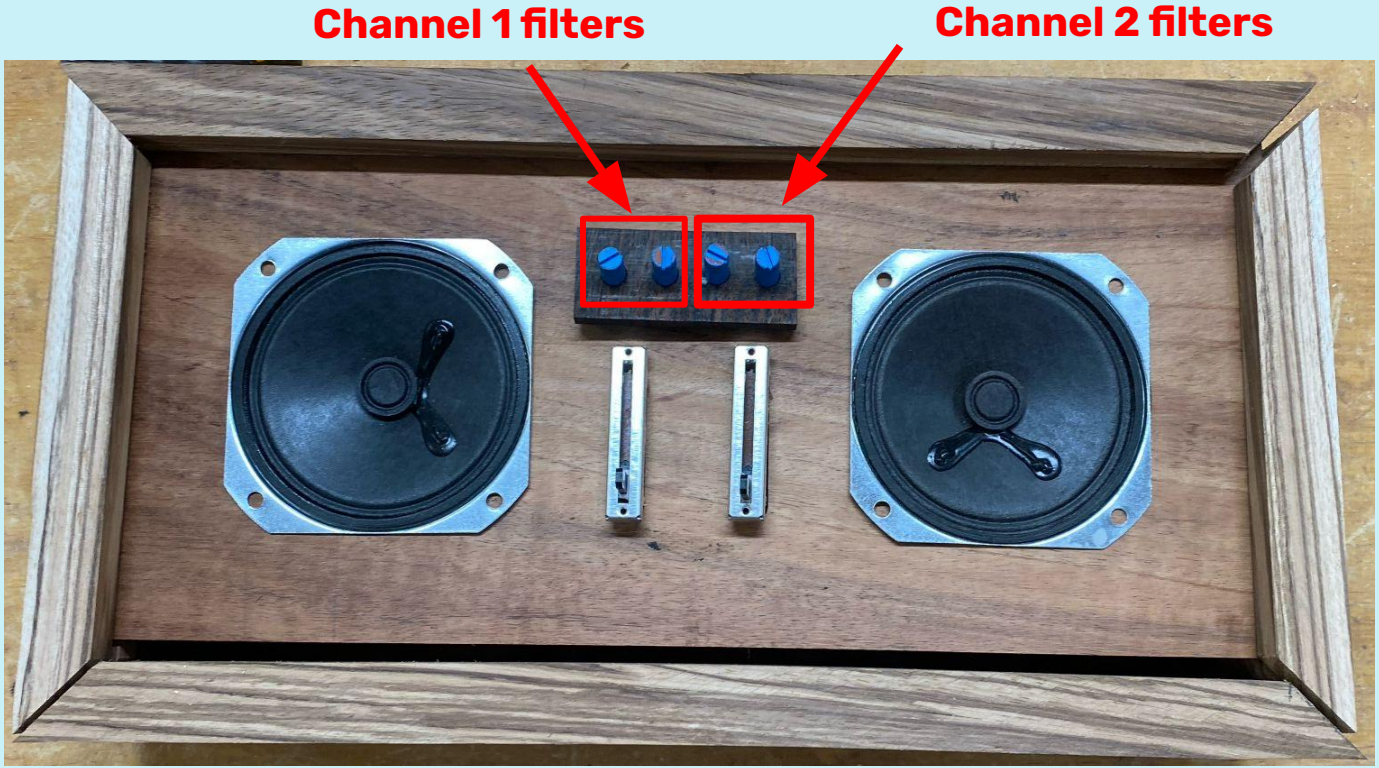
$$= \frac{80MHz}{10} = 8MHz$$

$$ARR = 9$$

*Data transfer is triggered by a reload event of Timer 7, which occurs at **800 kHz.***

Filtering

Reading User Inputs - MCU ADC



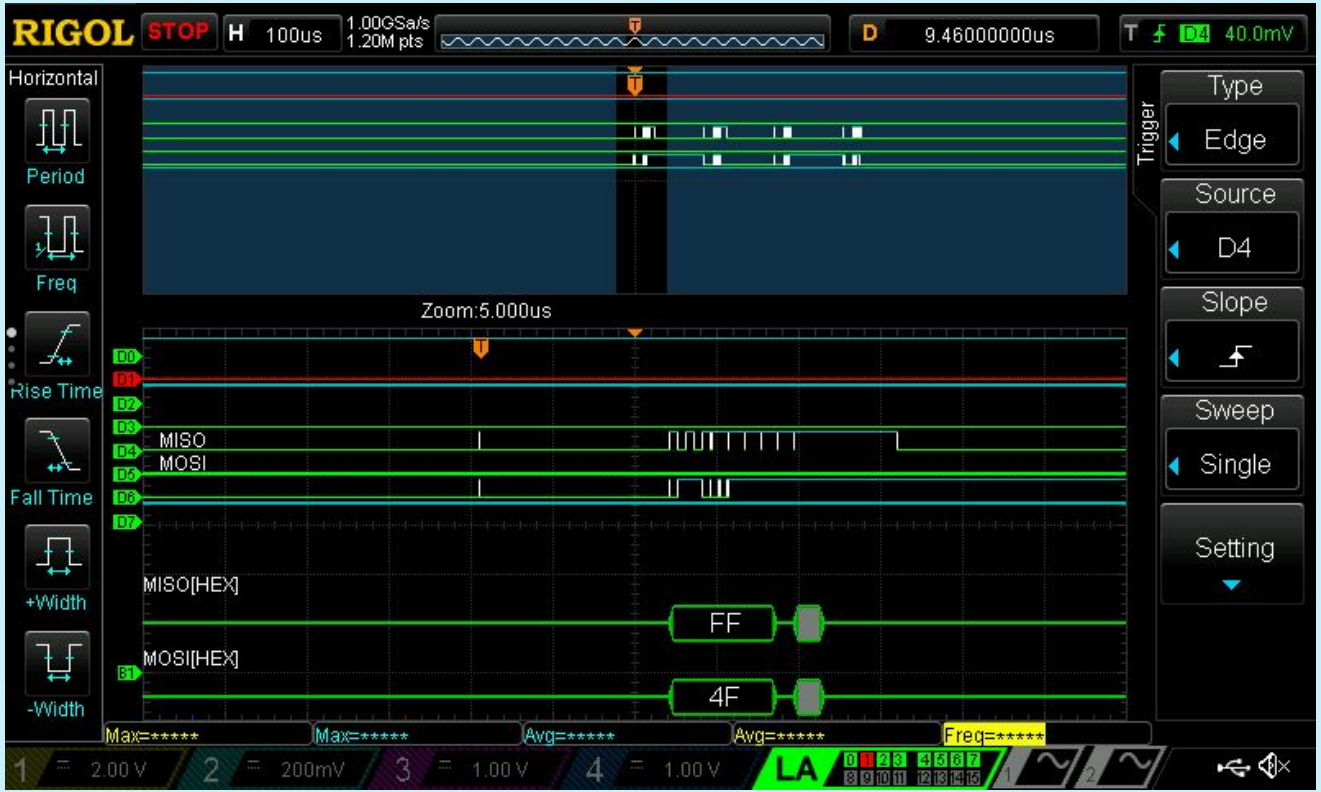
DJ Mixer Potentiometer Values

Confirming EQ Values

Debug Terminal

```
3rd channel: 81
4th channel: 78
1st channel: 79
2nd channel: 76
3rd channel: 81
4th channel: 78
1st channel: 79
2nd channel: 76
3rd channel: 81
4th channel: 78
```

Mapped EQ values & sample SPI interaction



Digital Signal Processing

FIR vs IFR

Finite Impulse Response Filter

$$y[n] = \sum_{k=0}^{N-1} h[k]\, x[n-k]$$

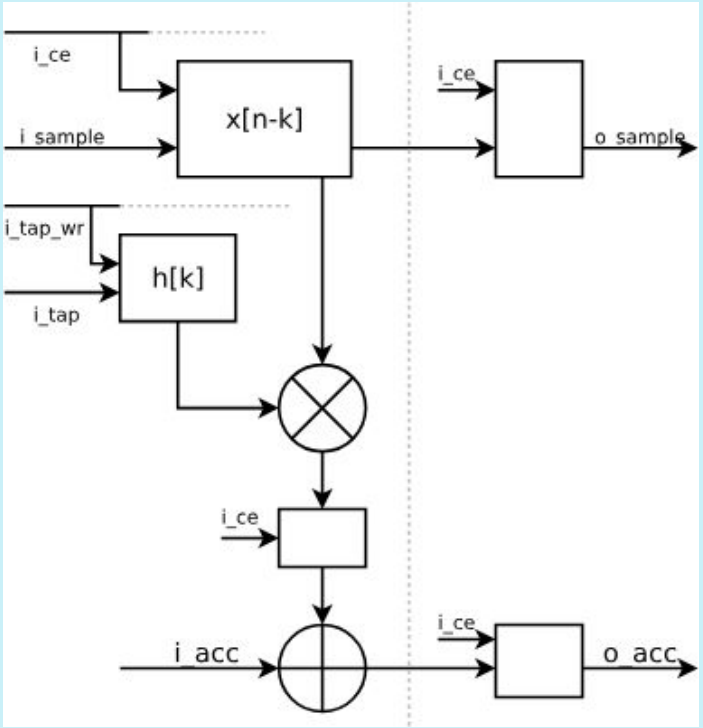
- Finite duration unit samples dependent on N
- Dependent on past & present inputs
- Does not require feedback

Infinite Impulse Response Filter

$$y[n] = bx[n] + ay[n-1],$$

- Infinite duration unit sample response
- Dependent on past & present inputs and past outputs
- Requires feedback

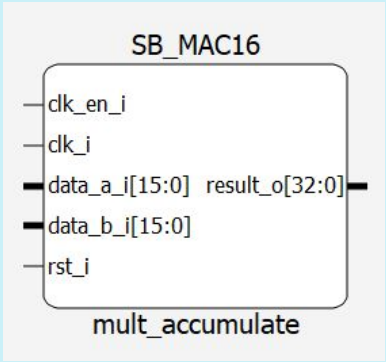
Filtering – FIR Filters



Block Diagram for one filter tap computation

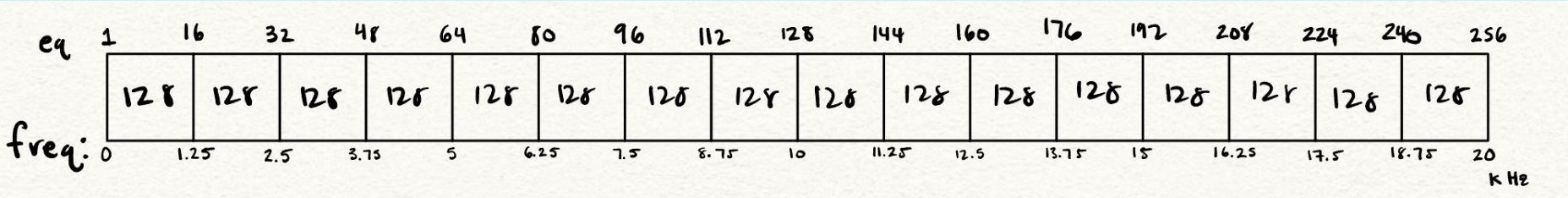
$$y[n] = \sum_{k=0}^{N-1} h[k]\, x[n-k]$$

By stringing together filter tap computations, we are able to apply an FIR filter

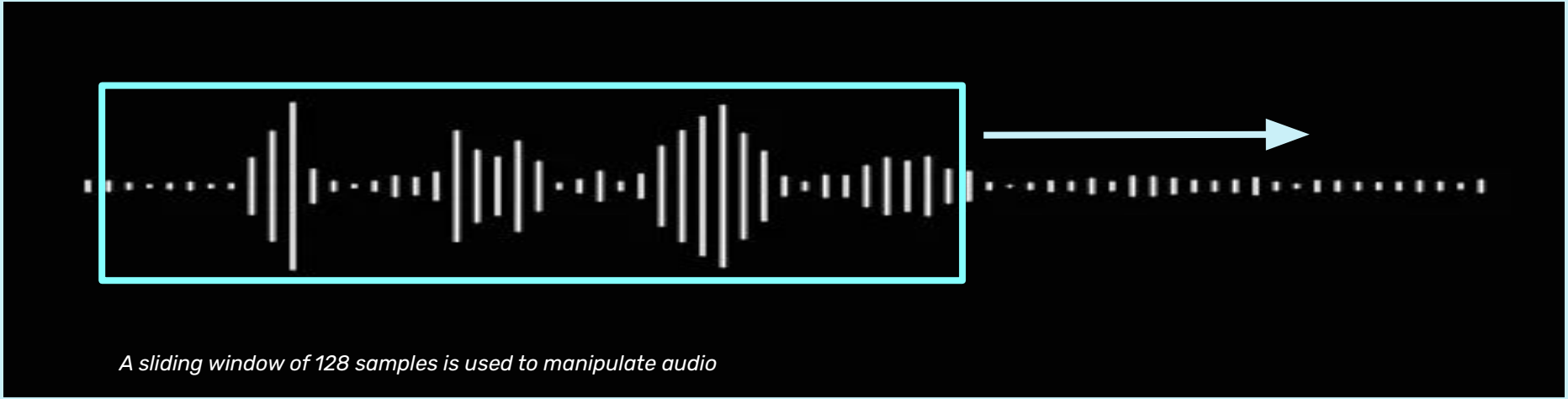


End goal was to leverage the Mult_accumulate block

Buckets: Our Implementation

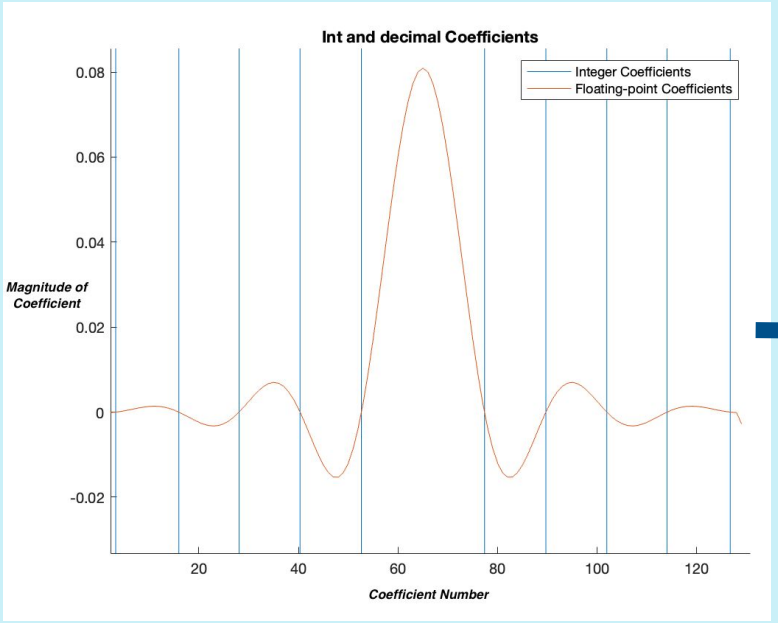


EQ Values mapped to buckets



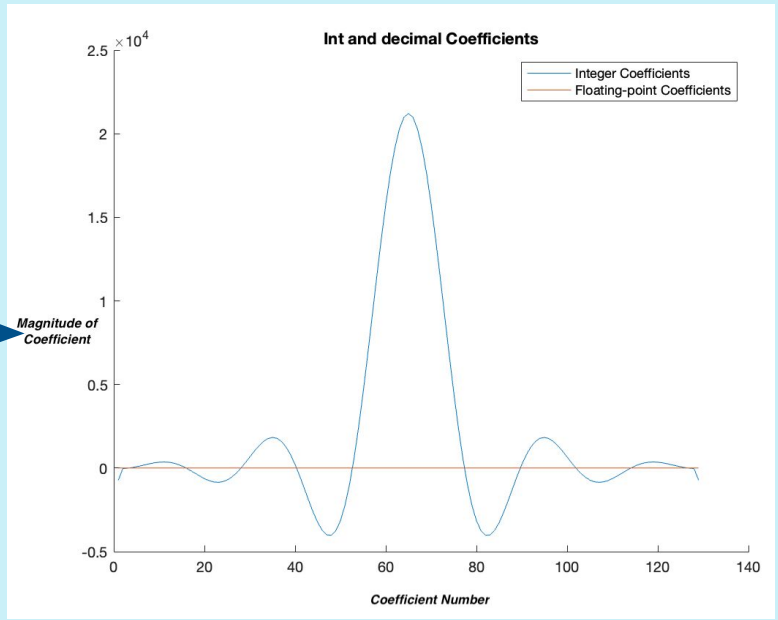
A sliding window of 128 samples is used to manipulate audio

FIR Matlab



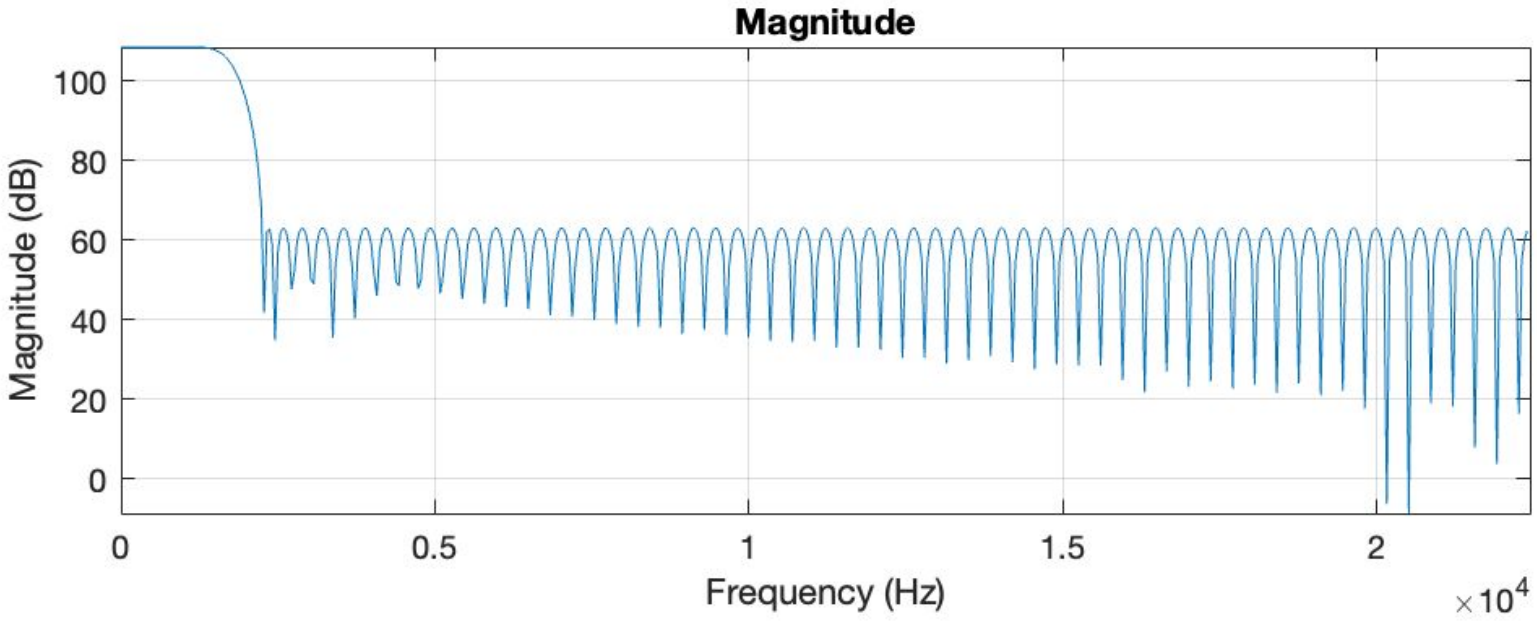
Coefficients as floating point values

Zooming into magnitude



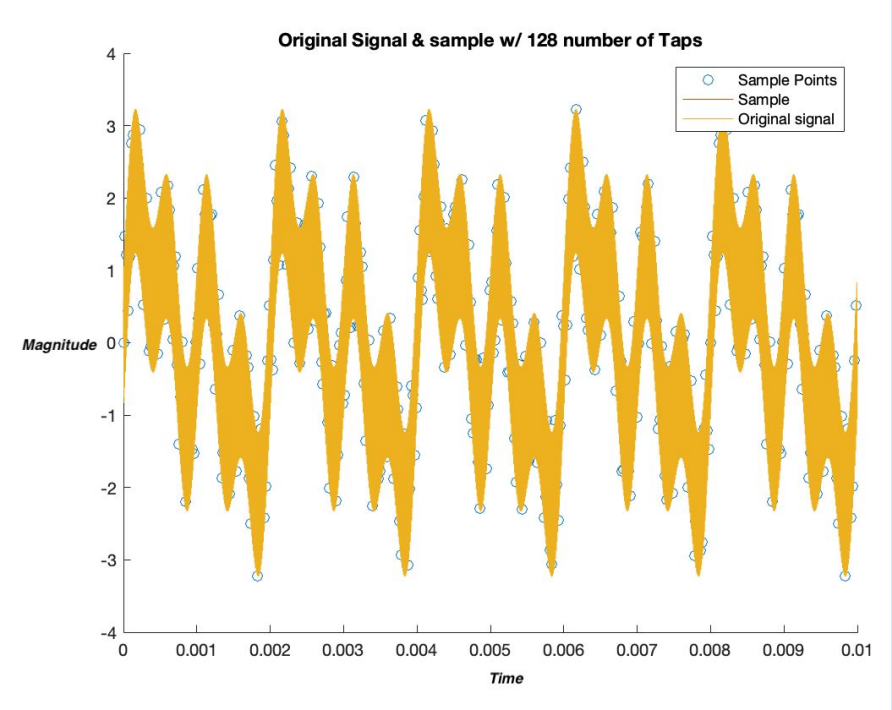
Coefficients as integers, scaled by g

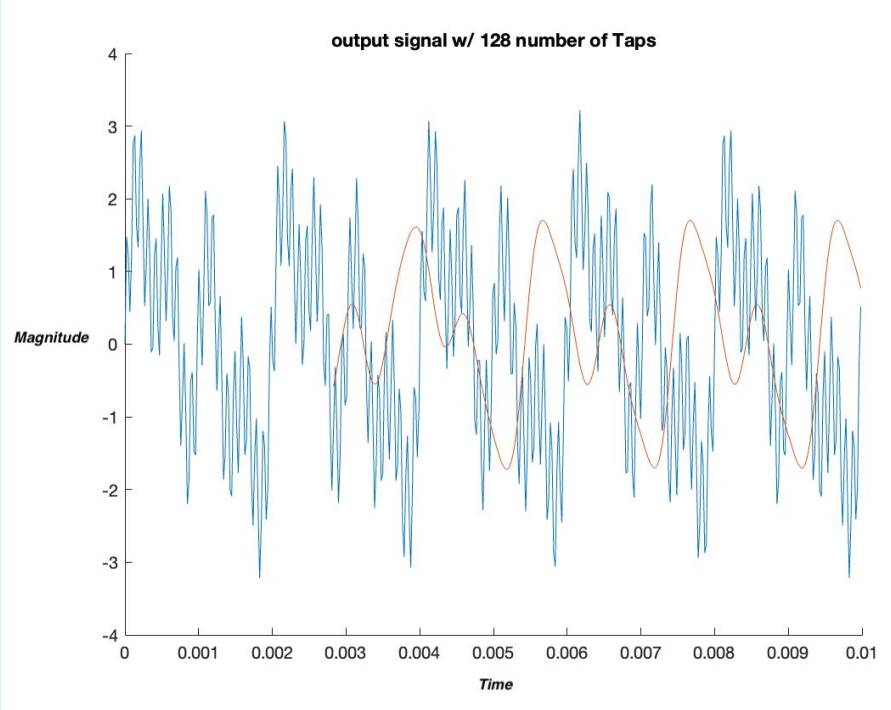
FIR Matlab



Sample lowpass filter with 128 taps

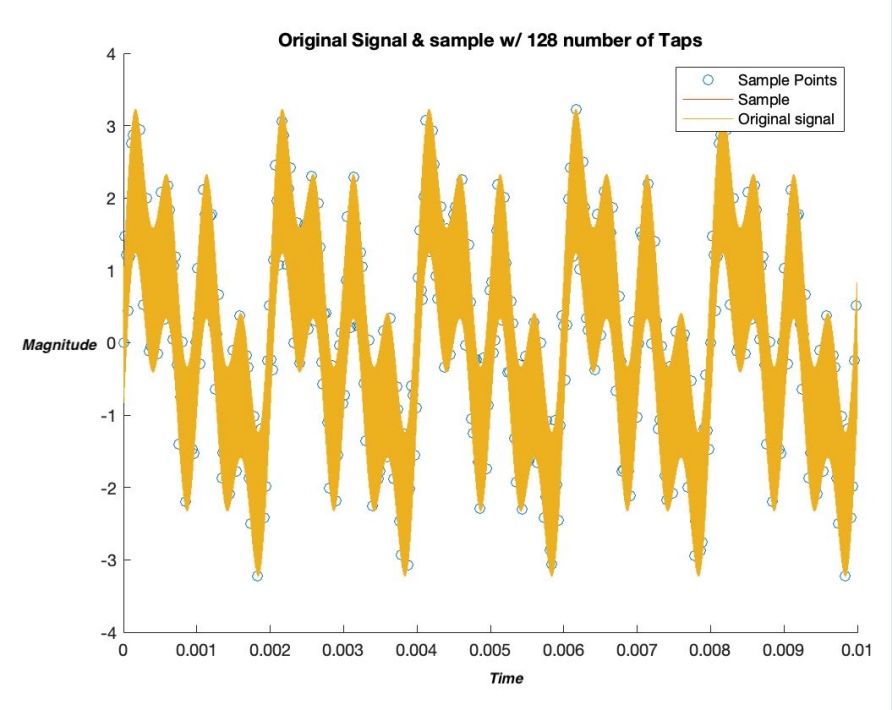
FIR Filters Implementation

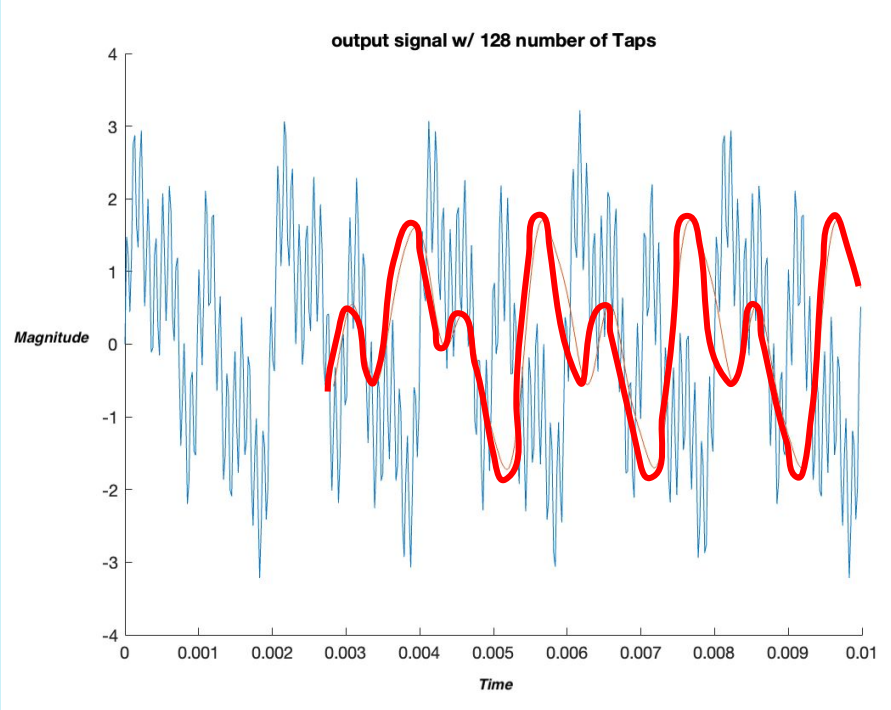




When applying FIR filters, we are able to take out high frequencies

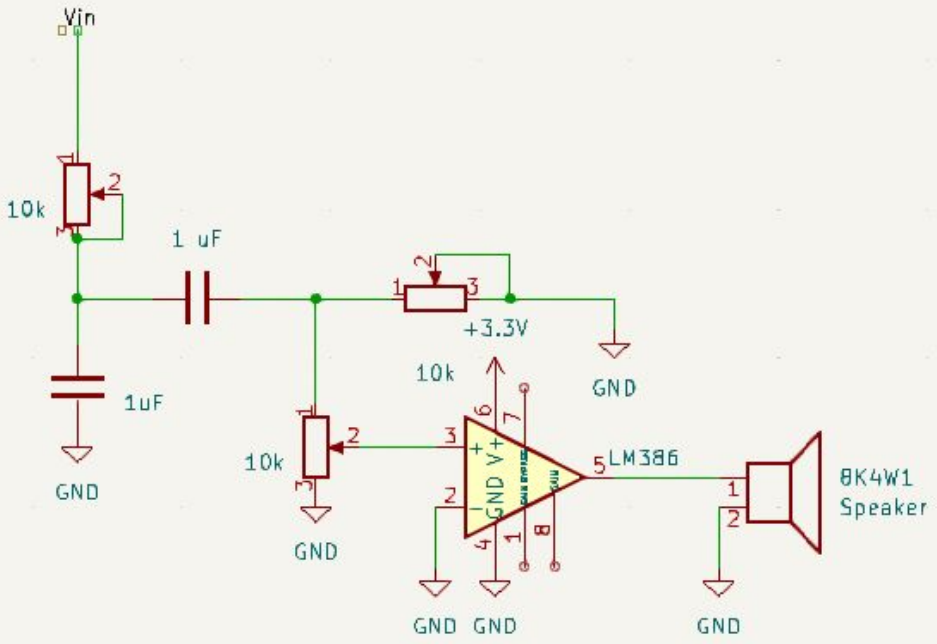
FIR Filters Implementation





When applying FIR filters, we are able to take out high frequencies

Pivot to Analog



RC/CR Circuit Diagram

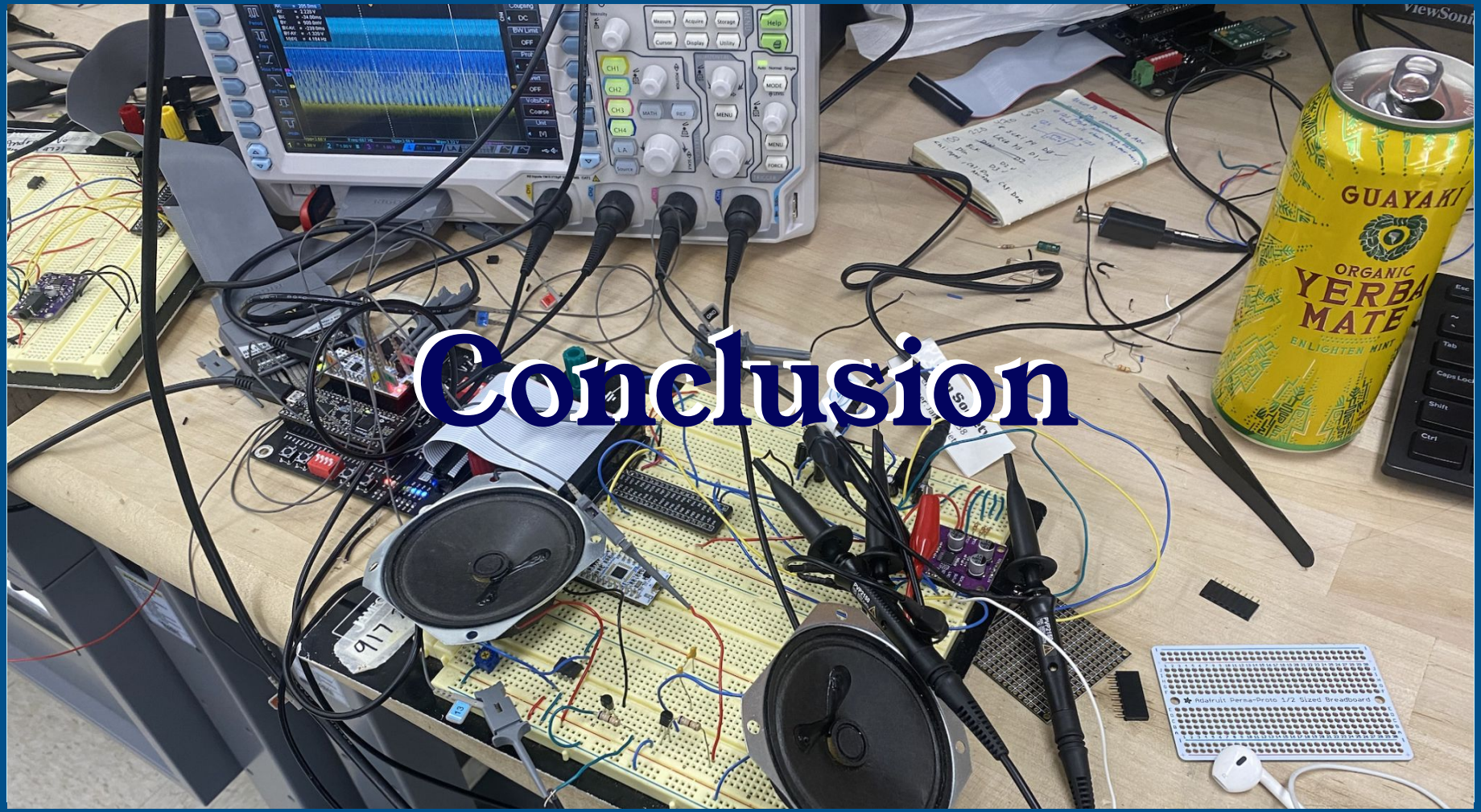
$$\text{Cutoff Frequency for RC/CR filter} = \frac{1}{2\pi rc}$$

$$\text{Potentiometer Range} = 5Hz to 10kHz$$

$$f_c = \frac{1}{2\pi rc} = \frac{1}{2\pi(10000)(1uF)} = 15.92Hz$$

$$f_c = \frac{1}{2\pi rc} = \frac{1}{2\pi(5)(1uF)} = 31,831Hz$$

Cutoff Frequency Calculations


GUAYAKI
ORGANIC
YERBA
MATE
ENLIGHTEN MINT
Adafruit Perma-Proto 1/2 Sized Breadboard

Conclusion

Debugging Techniques

A Summary of Bugs & Solutions

- Two’s Complement error
  - Signage error between FPGA and MCU
- SPI Timing
  - Too slow and overwriting samples
- Double checking logic in simulation
  - ModelSim & MatLab

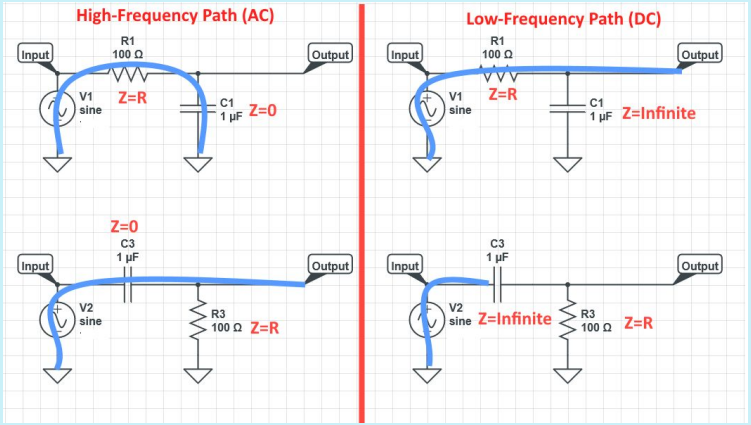
Two’s Complement error on visualized with oscilloscope

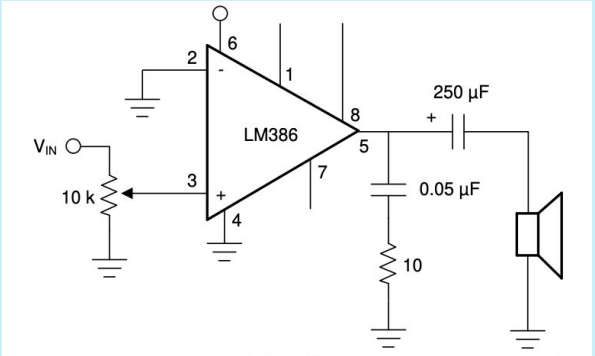
Results

System Characterization

- ✓ **Use external ADC to read in audio file**
  - System can read & output frequencies < 46.875 kHz
- ✓ **Communicate audio data from FPGA to MCU**
  - With SPI
- ✓ **Implement low and high pass digital filters**
  - FIR filters in simulation, RC/CR in hardware
- ✓ **Vary audio gain based on user inputs**
  - With potentiometer sliders
- ✓ **Output manipulated audio onto speaker**
  - Using STM DMA & DAC

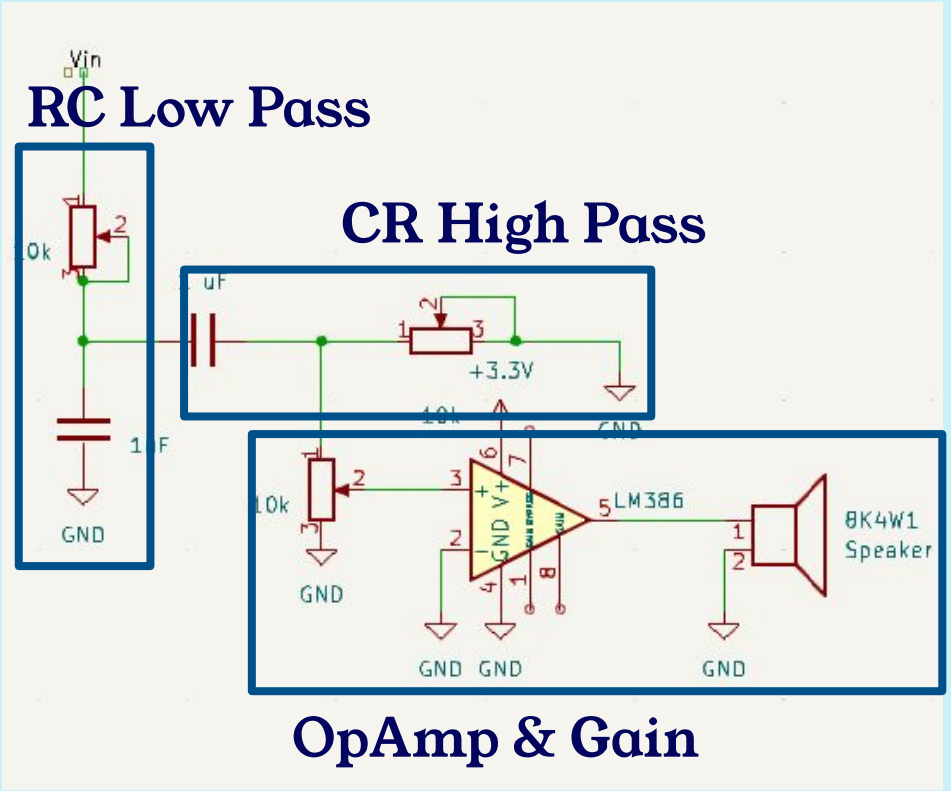

Questions?
SAN FRANCISCO
1000
NOTEL
MOTEL



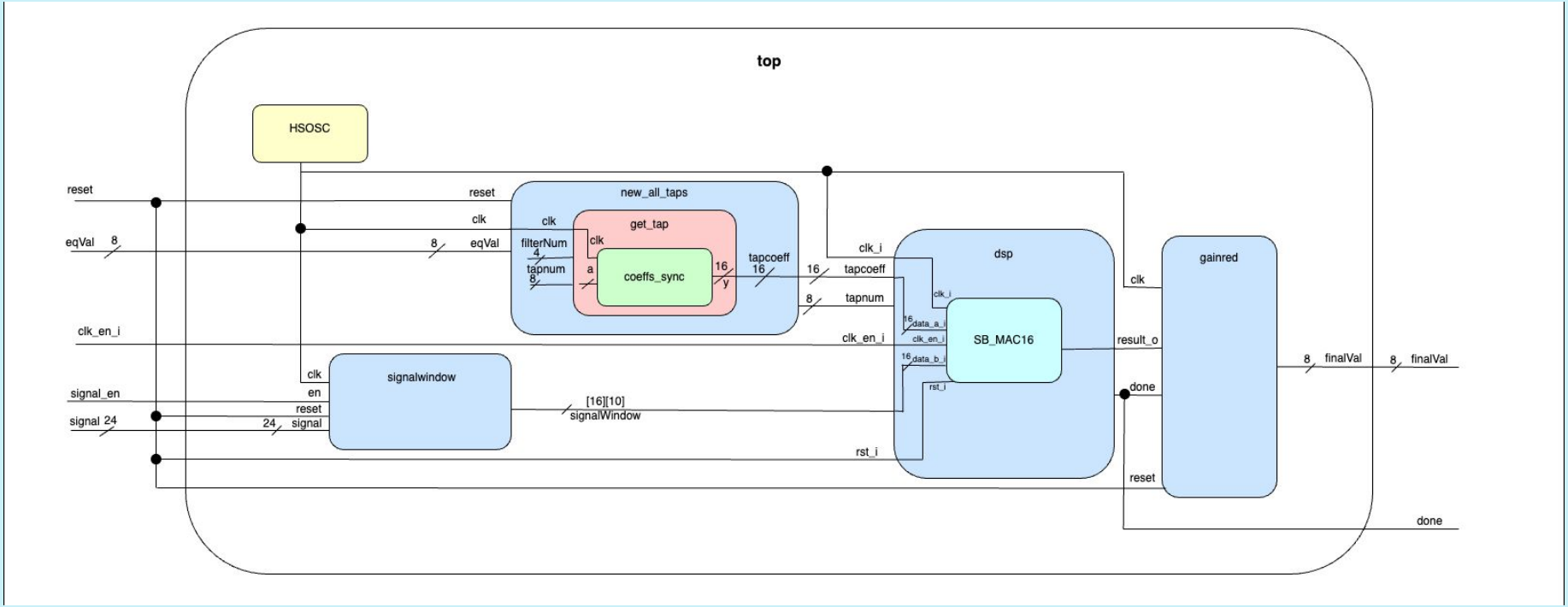


Copyright © 2017, Texas Instruments Incorporated

Figure 9-1. LM386 with Gain = 20

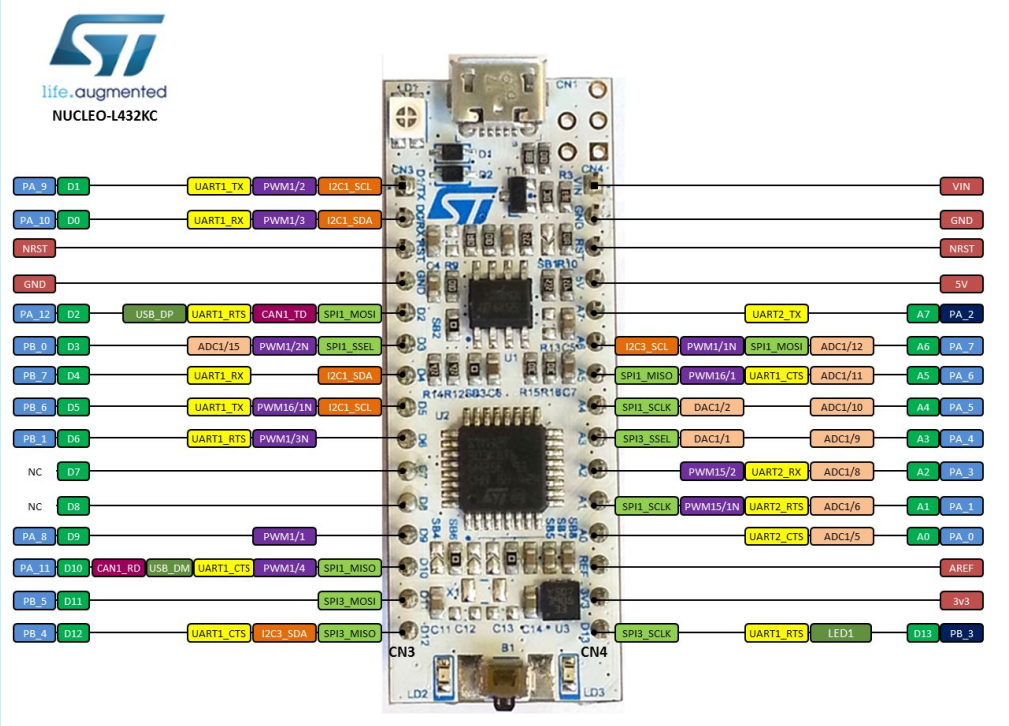


ModelSim



ModelSim Block Diagram of FIR filters

STM32L432KC



STM32L432KC Pin Diagram

STM32L4 MCU

- Maximum configurable clock frequency of up to 80 MHz
- 256 KB single bank Flash
- 1x 12-bit ADC 5 Msps
- 2x 8 or 12-bit DAC output channels
- 1x SAI, 2x SPI Communication interfaces
- 14-channel DMA controller

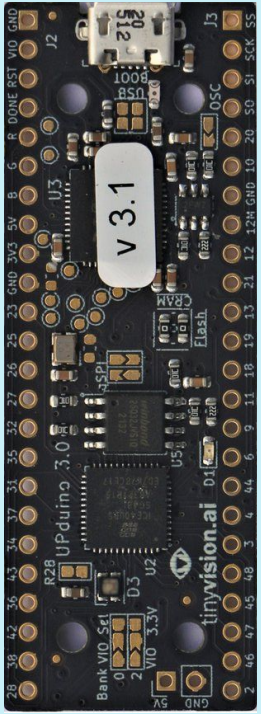
MCU Clock to 80 MHz

```
// Configure flash and PLL at 80 MHz
configureFlash();
configureClock();
```

```
void configureClock(){
  // Configure and turn on PLL
  configurePLL();

  // Select PLL as clock source
  RCC->CFGR = RCC_CFGR_SW_PLL | (RCC->CFGR & ~RCC_CFGR_SW);
  while((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);

  SystemCoreClockUpdate();
}
```

```
void configurePLL() {
   // Set clock to 80 MHz
   // Output freq = (src_clk) * (N/M) / P
   // (4 MHz) * (80/2) * 2  = 80 MHz
   // M:, N:, P:
   // Use HSI as PLLSRC

   RCC->CR &= ~_FLD2VAL(RCC_CR_PLLON, RCC->CR); // Turn off PLL
   while (_FLD2VAL(RCC_CR_PLLRDY, 1) != 0); // Wait till PLL is unlocked (e.g., off)

   // Load configuration
   RCC->PLLCFGR |= _VAL2FLD(RCC_PLLCFGR_PLLSRC, RCC_PLLCFGR_PLLSRC_MSI);
   RCC->PLLCFGR |= _VAL2FLD(RCC_PLLCFGR_PLLM, 0b001); // M = 2
   RCC->PLLCFGR |= _VAL2FLD(RCC_PLLCFGR_PLLN, 80);    // N = 80
   RCC->PLLCFGR |= _VAL2FLD(RCC_PLLCFGR_PLLR, 0b00);  // R = 2
   RCC->PLLCFGR |= RCC_PLLCFGR_PLLREN;                // Enable PLLCLK output

   // Enable PLL and wait until it's locked
   RCC->CR |= RCC_CR_PLLON;
   while(_FLD2VAL(RCC_CR_PLLRDY, RCC->CR) == 0);
}
```

UPduino v3.1 & iCE 40 FPGA



- **Clock speed**
- **Storage**
- **used lattice**
- **x LUTS**
- **x oscillator**
-

Configuring & Enabling

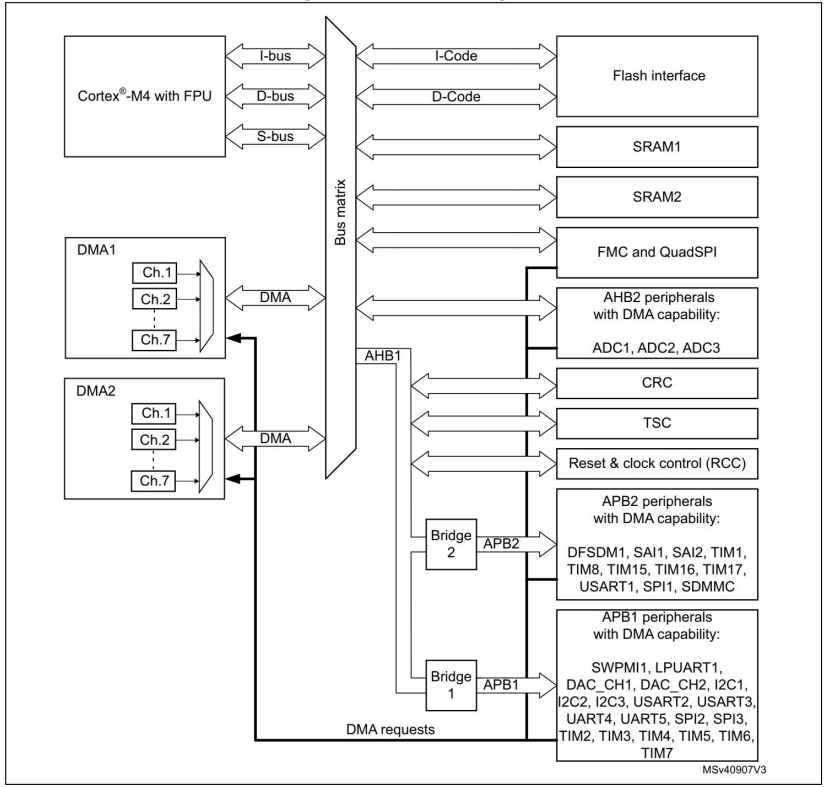
```
int main(void) {
  // Configure flash and PLL at 80 MHz
  configureFlash();
  configureClock();

  //// Enable interrupts
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
  __enable_irq();

  // ENABLE PERIPHERALS: DMA1, DMA2, DAC1, TIM7
  RCC->AHB1ENR |= (RCC_AHB1ENR_DMA1EN);   // Enable DMA 1
  RCC->APB1ENR1 |= RCC_APB1ENR1_DAC1EN; // Enable DAC1
  RCC->AHB1ENR |= (RCC_AHB1ENR_DMA2EN);   // Enable DMA 2
  RCC->APB1ENR1 |= (RCC_APB1ENR1_TIM7EN); // Enable TIM7

  /// Setup GPIO, Enable Clock for GPIO Ports A, B, and C
  gpioEnable(GPIO_PORT_A);
  gpioEnable(GPIO_PORT_B);
  gpioEnable(GPIO_PORT_C);

  // Set up analog input
  pinMode(ANALOG_IN1, GPIO_ANALOG);
  pinMode(ANALOG_IN2, GPIO_ANALOG);
  pinMode(ANALOG_IN3, GPIO_ANALOG);
  pinMode(ANALOG_IN4, GPIO_ANALOG);

  // Load and done pins
  pinMode(PA9, GPIO_OUTPUT);  // LOAD
  pinMode(PA6, GPIO_INPUT);   // DONE
```

DMA Initialization

```
// Artificial chip select signal to allow 8-bit CE-based SPI decoding on the logic analyzers.
digitalWrite(SPI_CE, 1);

// Enable GPIOA clock
RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN | RCC_AHB2ENR_GPIOBEN | RCC_AHB2ENR_GPIOCEN);

// Set up SPI
initSPI(0b1,0,0);

// Init DMA for Audio #1 (left)
 DMA1_Channel3->CCR &= ~(0xffffffff); // reset values
 DMA1_Channel3->CPAR = (uint32_t) &(DAC1->DHR8R1); //Place DMA into DAC Outplacement
 DMA1_Channel3->CMAR = (uint32_t) &leftAudio; // set source address to character array buffer in memory.
 DMA1_Channel3->CNDTR = sizeof(leftAudio); // Set DMA data transfer length (# of samples).
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_PL, 0b10);  // set priority lvl to very high B)
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_DIR, 0b1); // read from memory, 0b1
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_CIRC, 0b1); // enable circular mode
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_MINC, 0b1); // memory increment mode enabled
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_MSIZE, 0b00); //Memory size set to 8 bits, data size of DMA transfer in memory
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_PSIZE, 0b00); // peripheral size = 8 bits, Defines the data size of each DMA transfer
 DMA1_CSELR->CSELR |= _VAL2FLD(DMA_CSELR_C3S, 0b0110); // set dma channel 3 to DAC1_channel1
 DMA1_Channel3->CCR |= _VAL2FLD(DMA_CCR_EN, 0b1);
```

DMA BLOCK DIAGRAM

Figure 27. DMA block diagram



LOAD & DONE on MCU

```
// Write LOAD high
digitalWrite(PA9, 1);

for(i = 0; i < 4; i++) {
  spiSendReceive((char)convertedVals[i]);
}
while(SPI1->SR & SPI_SR_BSY); // Confirm all SPI transactions are completed
digitalWrite(PA9, 0); // Write LOAD low


// Wait for DONE signal to be asserted by FPGA signifying that the data is ready to be read out.
while(!digitalRead(PA6));
leftAudioneg = spiSendReceive(0);
leftAudio = leftAudioneg + 128; // shift to get signed to unsigned (DAC cannot send negative voltages)
```

LOAD & DONE on FPGA

```
module eq1_core(input logic clk,
                        input logic [7:0] left,
                        //input logic [7:0] right,
                        input  logic         load,
            input  logic [31:0] eqVals,
            output logic         done,
            output logic [7:0] finalVal);

//logic [10:0] counter;
logic [7:0] leftTemp;
//logic [7:0] rightTemp;
always_ff @(posedge clk) begin
        if (load) begin
                done <= 0;
                //counter <= 0;
                leftTemp <= left;
                //rightTemp <= right;
    end
        else begin
                        finalVal <= leftTemp;//, rightTemp};
                        done <= 1;
                end
end
endmodule
```

Prescalers for Clock Config

```
module i2s(input logic          clk,
           input logic          reset,
           input logic          din, //  PCM1808 DOUT,          PB6_G12
           output logic         bck, //  bit clock,             PA7_J2
           output logic         lrck, // left/right clk,        PA6_J1
           output logic         scki, // PCM1808 system clock,  PA5_H4
           output logic [23:0] left,
           output logic [23:0] right);


  //   Fs = 92KkHz
  logic [8:0]                  prescaler; // 9-bit prescaler
  assign scki = clk;           // 256 * Fs = 24 Mhz
  assign bck  = prescaler[1]; // 64  * Fs
  assign lrck = prescaler[7]; // 1   * Fs

  always_ff @(posedge clk)
    begin
      if (reset)
        prescaler <= 0;
      else
        prescaler <= prescaler + 9'd1;
    end
```

*SCKI = 256 * Fs = 24 MHz*
*BCK = 64 * Fs = 6 MHz*
LRCK = Fs = 93.75 kHz

Can sample frequencies up to 46.875 kHz

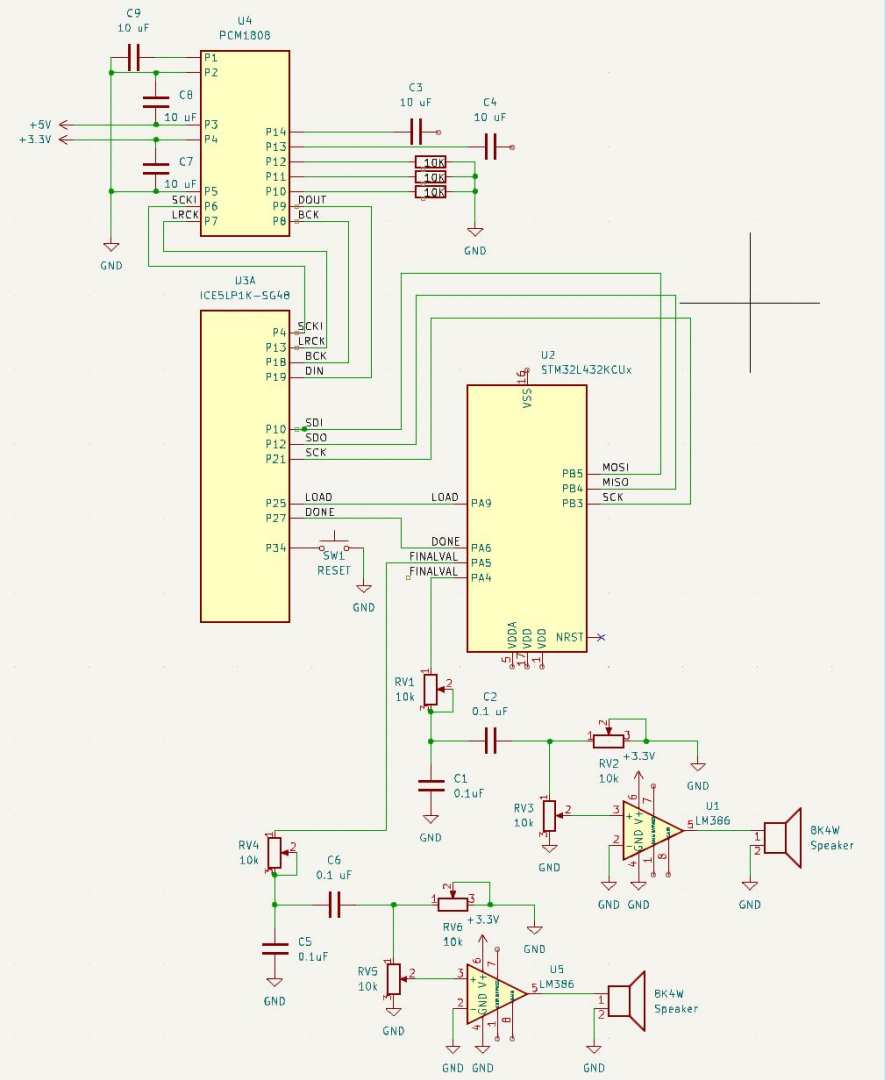
SPI Module FPGA

```
module eq1_spi(
                        input  logic sck,
              input  logic sdi,
                        output logic sdo,
              input  logic done,
              output logic [31:0] eqVals,
                        input logic [7:0] finalVal);

        logic sdodelayed, wasdone;
        logic [7:0] finalValCaptured;

        always_ff @(posedge sck) begin
        if (!wasdone)  {finalValCaptured, eqVals} = {finalVal, eqVals[30:0], sdi};
        else   begin
                        {finalValCaptured, eqVals} = {finalValCaptured[6:0], eqVals, sdi};
                end
    end

    // sdo should change on the negative edge of sck
    always_ff @(negedge sck) begin
        wasdone = done;
        sdodelayed = finalValCaptured[6];
    end

    // when done is first asserted, shift out msb before clock edge
    assign sdo =  (done & !wasdone) ? finalVal[7] : sdodelayed;

endmodule
```

TOP Module Declaration

```
module top(input logic nreset,
                  input logic din, // I2S DOUT, P19

                  input  logic sck, // SPI!! from MCU P21 for SPI
           input  logic sdi, //  SPI!! mosi

                  output logic       bclk, //     I2S bit clock,     P18
           output logic       lrck, //    I2S l/r clk,       P13
           output logic       scki,

           output logic sdo, // SPI!!
           input  logic load, // SPI!!
                  output logic done  // SPI!!
                 );

    // assign reset
        logic reset;
        assign reset = ~(nreset);

        logic clk;
    HSOSC #(.CLKHF_DIV ("0b01")) hf_osc (.CLKHFPU(1'b1), .CLKHFEN(1'b1), .CLKHF(clk));  // set divider to 0b01 get 24MHz clock
```


RIGOL


C9
10 uF
U4
PCM1808
P1
P2
C8
10 uF
P3
P4
C7
10 uF
P5
P6
P7
SCKI
LRCK
P14
P13
P12
P11
P10
P9
P8
DOUT
BCK
C3
10 uF
C4
10 uF
10K
10K
10K
+5V
+3.3V
GND
GND
U3A
ICE5LP1K-SG48
P4
P13
P18
P19
SCKI
LRCK
BCK
DIN
P10
P12
P21
SDI
SDO
SCK
P25
P27
P34
LOAD
DONE
SW1
RESET
GND
U2
STM32L432KCUx
VSS
PB5
PB4
PB3
MOSI
MISO
SCK
LOAD
PA9
DONE
FINALVAL
FINALVAL
PA6
PA5
PA4
VDDA
VDD
VDD
NRST
RV1
10k
C2
0.1 uF
C1
0.1uF
GND
RV2
10k
+3.3V
GND
RV3
10k
GND
U1
LM386
GND
GND
8K4W
Speaker
GND
RV4
10k
C6
0.1 uF
C5
0.1uF
GND
RV6
10k
+3.3V
GND
RV5
10k
GND
U5
LM386
GND
GND
8K4W
Speaker
GND

FPGA Block Diagram