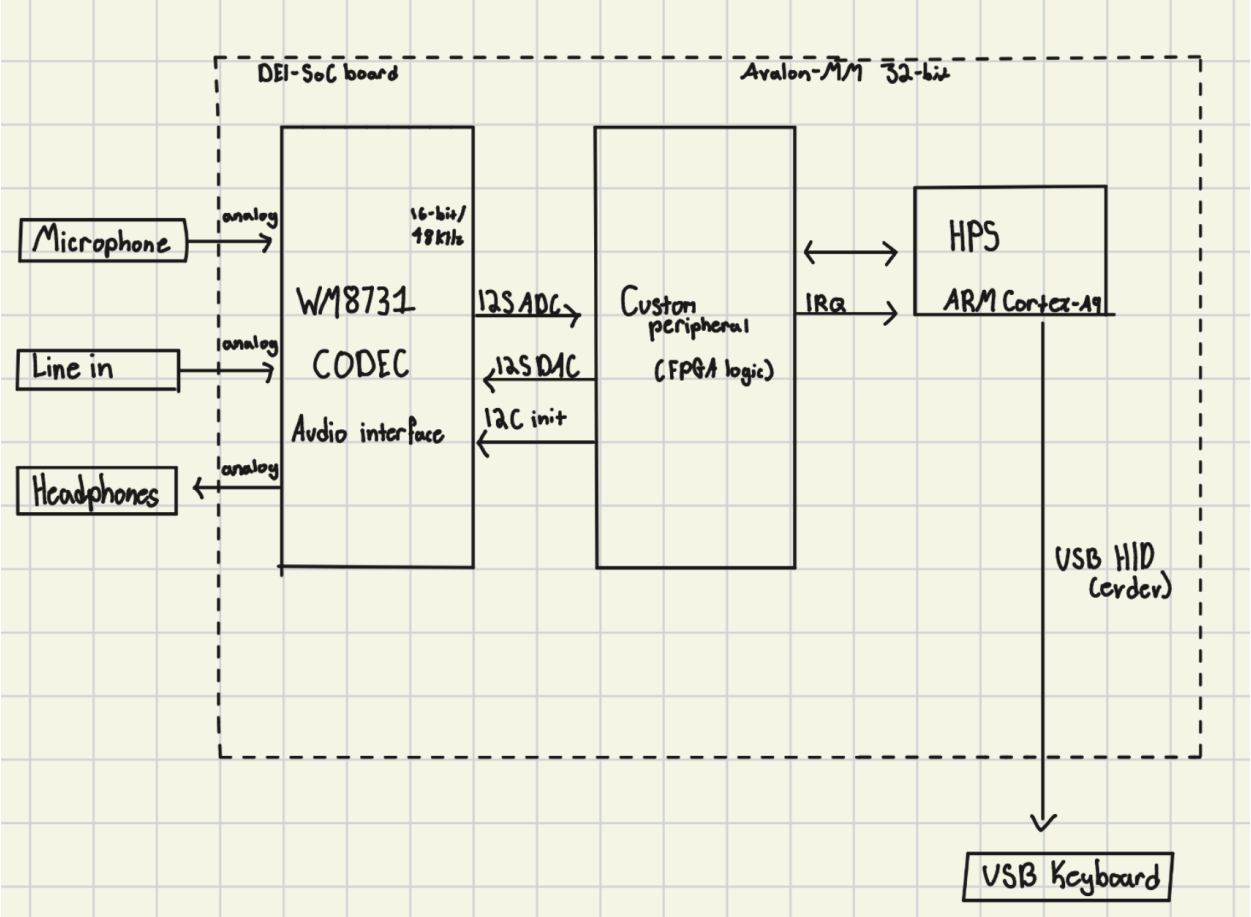
Adaptive Noise Cancellation System Design Document
Huda Jafri (shj2127), Sharvani Vadlamani (sv2734), Sayem Kamal (sk5336)

Project Overview:

We intend to implement a real-time hardware adaptive noise cancellation system on the DE1-SoC development board. This system will accept two simultaneous audio inputs through the onboard Wolfson WM8731 audio CODEC. The first of these will be a microphone signal of speech mixed with background noise, and the second will be a reference signal with only the noise source. A hardware-accelerated LMS adaptive FIR filter running on the FPGA will estimate and subtract the noise component from the microphone signal, outputting clean audio to the headphone jack with minimal latency.

The time-critical audio processing loop will run entirely on programmable logic. The HPS (ARM processor) will handle only non-time-critical tasks, such as reading the keyboard input and updating the filter parameters over the Avalon bus.

Top-Level Block Architecture:



Connection Protocols:

Microphone / Line In / Headphones → WM8731 CODEC (analog):
Standard 3.5 mm jack connections to the CODEC's Mic In, Line In, and Line Out pins. No digital protocol; the CODEC's ADC and DAC handle conversion.

WM8731 CODEC – Custom Peripheral (I2C, 400 kHz — configuration only):
2-wire open-drain bus (FPGA_I2C_SDAT / FPGA_I2C_SCLK). The i2c_controller inside the peripheral sends a fixed sequence of 16 three-byte write transactions at startup: [0x34 | W] [reg_addr[6:1] | reg_addr[0]] [data[8:1] | data[0]]. After all transactions complete, the controller asserts init_done and goes idle for the rest of operation.

WM8731 CODEC – Custom Peripheral (I2S, 48 kHz — real-time audio):
4-wire I2S: BCLK (3.072 MHz bit clock from CODEC), LRCLK / ADCLRCK / DACLRCK (48 kHz frame clock), ADCDAT (serial input to FPGA), DACDAT (serial output from FPGA). Data is MSB-first, 16 bits per frame half. LRCLK LOW = left channel (Line In → reference x[n]); LRCLK HIGH = right channel (Mic In → desired signal d[n]). The FPGA samples ADCDAT on BCLK falling edges and drives DACDAT on BCLK rising edges.

Custom Peripheral – HPS (Avalon-MM, 32-bit — parameter control):
Memory-mapped interface on the lightweight HPS-to-FPGA bridge. HPS drives a 5-bit word address, read/write strobes, and 32-bit write data. Peripheral returns 32-bit read data in the same cycle (zero wait states). An IRQ line from the peripheral to the HPS GIC fires when sample_ready is set and IRQ_ENABLE[0] = 1. The HPS clears the interrupt by writing 1 to STATUS[0].

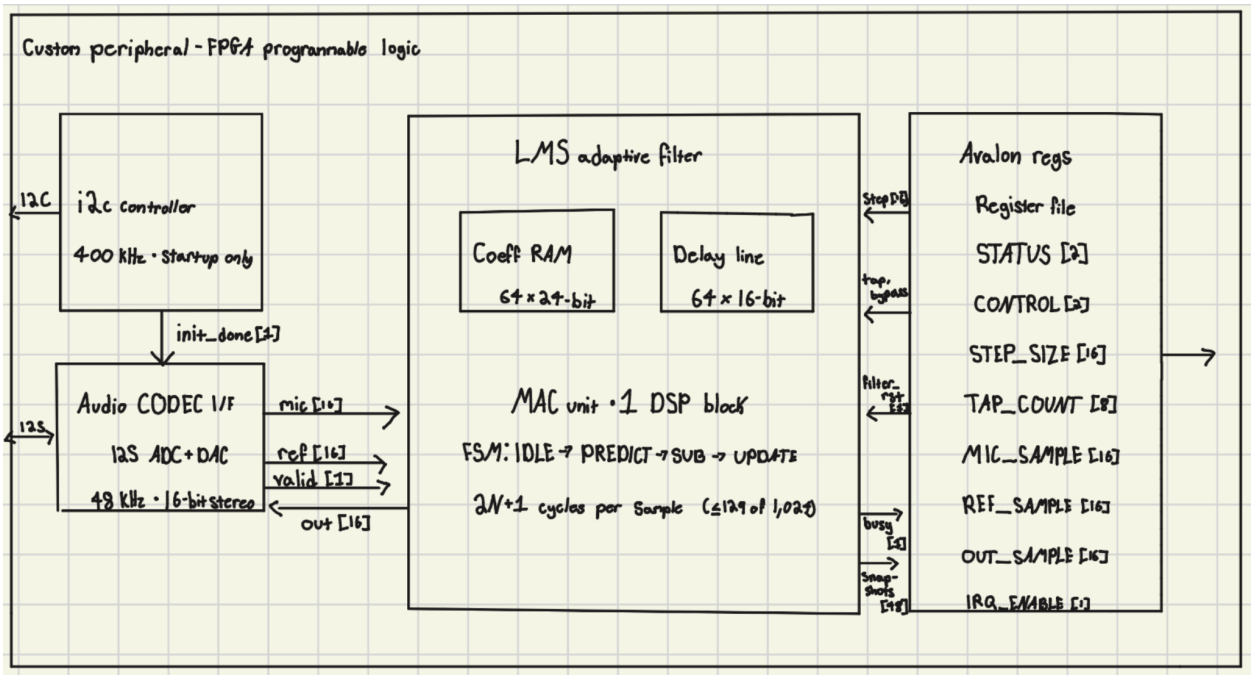
HPS – USB Keyboard (USB HID via Linux evdev):
The keyboard enumerates as a USB HID device on the HPS USB OTG port. Linux exposes key events through /dev/input/eventX. Software reads struct input_event records (type = EV_KEY, value = 1 for press). No FPGA logic is involved; key presses trigger writes to the peripheral register map via /dev/mem.

Register Map

Note that all registers are 32-bit wide and unused bits are read as 0 and ignored on write.

Offset	Name	R/W	Desciption
0x00	STATUS	R/W	Bit 0 (sample_ready): set by hardware on each new 48 kHz audio sample; cleared by writing 1 to this bit. Bit 1 (filter_busy): read-only; high while LMS FSM is in PREDICT or UPDATE phase. Writing to STATUS[1] is a no-op.
0x04	CONTROL	R/W	Bit 0 (bypass): when set, audio_out = mic_in unmodified; LMS filter is held idle. Bit 1 (filter_reset): writing 1 zeros all 64 LMS coefficients; self-clears after one clock cycle and reads back as 0.
0x08	STEP_SIZE	R/W	Bits [15:0]: LMS step size μ in Q0.16 unsigned fixed-point. Range 0x0000 (no adaptation) to 0xFFFF (≈0.999). Default 0x0100 (≈0.004). Takes effect on the next audio sample after the write. Larger values converge faster but risk instability.
0x0C	TAP_COUNT	R/W	Bits [7:0]: number of active LMS filter taps. Valid range 1–64; writes outside this range are silently ignored. Default 32. Takes effect on the next audio sample.
0x10	MIC_SAMPLE	R	Bits [15:0]: snapshot of the most recent microphone input d[n] in Q1.15 signed format. Latched simultaneously with REF_SAMPLE and OUT_SAMPLE on each sample_ready pulse. Writing is a no-op.
0x14	REF_SAMPLE	R	Bits [15:0]: snapshot of the most recent reference noise input x[n] in Q1.15 signed format. Latched simultaneously with MIC_SAMPLE and OUT_SAMPLE. Writing is a no-op.
0x18	OUT_SAMPLE	R	Bits [15:0]: snapshot of the most recent filter output e[n] in Q1.15 signed format. Latched simultaneously with MIC_SAMPLE and REF_SAMPLE. Writing is a no-op.
0x1C	IRQ_ENABLE	R/W	Bit 0 (irq_en): when set, a hardware interrupt is sent to the HPS GIC on each new audio sample (each time sample_ready is set). Default 0 (polling mode). Clear to use polling via STATUS[0] instead.

Custom Peripheral Internal Block Diagram:



Memories:

Memory	**DepthxWidth**	**Type**	**Purpose**
Coefficient RAM	64x24 bit	M10K (dual-port)	Stores LMS filter weights w[k]. Port A reads coefficients during PREDICT; Port B writes updated coefficients during UPDATE.
Reference delay line	64x16 bit	Registers (circular buf.)	Stores recent x[n] samples needed for both FIR convolution and coefficient update. Updated by shifting in the new x[n] on each sample_valid.
I2C command ROM	16x25 bit	Registers (read-only)	Fixed sequence of WM8731

			register-write commands: {dev_addr[6:0], reg_addr[6:0], data[8:0]}. Read sequentially by i2c_controller at startup.

Blocks and Connections:

From	To	Width	Signal & Protocol
audio_codec_interface	lms_filter	16	mic_sample: Q1.15 PCM sample d[n], valid when sample_valid = 1
audio_codec_interface	lms_filter	16	ref_sample: Q1.15 PCM sample x[n], valid when sample_valid = 1
audio_codec_interface	lms_filter, avalon_regs	1	sample_valid: single-cycle strobe at 48 kHz. Upstream (audio interface) drives it high for exactly one 50 MHz clock cycle. Downstream modules latch their inputs on this pulse and begin processing
lms_filter	audio_codec_interface	16	audio_out: Q1.15 filtered output e[n]. Held stable from end of SUBTRACT phase until next sample_valid.

			audio_codec_interface latches this when it needs to serialize the DAC frame.
lms_filter	avalon_regs	1	busy: high during PREDICT and UPDATE phases. Exposed as STATUS[1].
i2c_controller	audio_codec_interface	1	init_done: held high after all 16 I2C transactions complete. Gates the audio_codec_interface enable; I2S processing does not begin until this is asserted
avalon_regs	lms_filter	16	step_size: registered copy of STEP_SIZE[15:0]. Updated by HPS write; takes effect next sample
avalon_regs	lms_filter	8	tap_count: registered copy of TAP_COUNT[7:0
avalon_regs	lms_filter	1	bypass: registered copy of CONTROL[0]. When high, lms_filter passes mic_sample to audio_out unchanged and skips PREDICT/UPDATE

avalon_regs	lms_filter	1	filter_reset: single-cycle pulse generated when HPS writes 1 to CONTROL[1]. Causes lms_filter to zero all coefficient RAM entries.

Verilog Module Interfaces:
noise_cancel_top: top-level wrapper that instantiates: i2c_controller, audio_codec_interface, lms_filter, avalon_regs.

```
None
module noise_cancel_top (
  input  wire        CLOCK_50,         // 50 MHz system clock
  input  wire        reset_n,          // Active-low reset (KEY[0])
  // Audio CODEC pins
  input  wire        AUD_BCLK,         // 3.072 MHz bit clock from CODEC
  input  wire        AUD_DACLRCK,      // 48 kHz DAC LR clock
  input  wire        AUD_ADCLRCK,      // 48 kHz ADC LR clock
  input  wire        AUD_ADCDAT,       // Serial ADC data (in to FPGA)
  output wire        AUD_DACDAT,       // Serial DAC data (out from FPGA)
  output wire        AUD_XCK,          // 12.288 MHz master clock to CODEC
  // I2C (CODEC configuration)
  inout  wire        FPGA_I2C_SDAT,    // Open-drain data
  output wire        FPGA_I2C_SCLK,   // 400 kHz clock
  // Avalon-MM slave (lightweight HPS-to-FPGA bridge)
  input  wire [4:0]  avs_address,      // 5-bit word address (32
registers)
  input  wire        avs_read,
  input  wire        avs_write,
  input  wire [31:0] avs_writedata,
```

```
  output wire [31:0] avs_readdata,
  output wire        avs_irq           // Level interrupt to HPS GIC
);
```

i2c_controller: contains internal command ROM (16 × 25-bit registers).

```
None
module i2c_controller (
  input  wire        clk,              // 50 MHz
  input  wire        rst_n,
  inout  wire        sda,              // Open-drain I2C data
  output wire        scl,              // 400 kHz clock (50 MHz / 125)
  output wire        init_done         // High after all 16 CODEC regs
written
);
```

audio_codec_interface: contains 16-bit ADC shift register, 16-bit DAC shift register, LRCLK edge detector, two-flop CDC synchronizer

```
None
module audio_codec_interface (
  input  wire        clk,              // 50 MHz
  input  wire        rst_n,
  input  wire        init_done,        // From i2c_controller; gates
enable
  // I2S pins
  input  wire        bclk,             // 3.072 MHz (async to clk)
  input  wire        adc_lrclk,        // 48 kHz; HIGH = right (mic)
  input  wire        dac_lrclk,
  input  wire        adcdat,
  output wire        dacdat,
  // Parallel sample interface (sync to clk)
  output reg  [15:0] mic_sample,       // d[n]: right channel (Mic In)
  output reg  [15:0] ref_sample,       // x[n]: left channel (Line In)
  input  wire [15:0] dac_sample,       // e[n]: drives DAC serializer
  output wire        sample_valid      // 1-cycle strobe at 48 kHz
);
```

lms_filter: contains coefficient RAM (64 × 24-bit M10K), delay line registers (64 × 16-bit), MAC unit (1 DSP block), 4-state FSM

None

```
module lms_filter #(
  parameter N            = 64,          // Max taps
  parameter DATA_WIDTH   = 16,
  parameter COEFF_WIDTH  = 24
) (
  input  wire                     clk,
  input  wire                     rst_n,
  input  wire [DATA_WIDTH-1:0]    mic_in,       // d[n]
  input  wire [DATA_WIDTH-1:0]    ref_in,       // x[n]
  input  wire                     sample_valid,
  input  wire [15:0]              step_size,    // mu, Q0.16
  input  wire [7:0]               tap_count,    // Active taps 1-64
  input  wire                     bypass,
  input  wire                     filter_reset,
  output wire [DATA_WIDTH-1:0]    audio_out,    // e[n]
  output wire                     busy
);
```

avalon_regs: contains 8 × 32-bit registers, sample snapshot latches, IRQ logic.

None

```
module avalon_regs (
  input  wire        clk,
  input  wire        rst_n,
  // Avalon-MM slave port
  input  wire [4:0]  address,          // Word address
  input  wire        read,
  input  wire        write,
  input  wire [31:0] writedata,
  output reg  [31:0] readdata,
  output wire        irq,
  // Hardware connections
  input  wire [15:0] mic_sample,
  input  wire [15:0] ref_sample,
  input  wire [15:0] out_sample,
  input  wire        sample_valid,
  input  wire        busy,
  output reg  [15:0] step_size,
  output reg  [7:0]  tap_count,
  output reg         bypass,
```

```
  output reg         filter_reset      // Self-clearing after 1 cycle
);
```

C Header File:

```C/C++
/* noise_cancel.h */
#ifndef NOISE_CANCEL_H
#define NOISE_CANCEL_H
#include <stdint.h>
/* Register offsets from NC_BASE_ADDR */
#define NC_OFF_STATUS      0x00
#define NC_OFF_CONTROL     0x04
#define NC_OFF_STEP_SIZE   0x08
#define NC_OFF_TAP_COUNT   0x0C
#define NC_OFF_MIC_SAMPLE  0x10
#define NC_OFF_REF_SAMPLE  0x14
#define NC_OFF_OUT_SAMPLE  0x18
#define NC_OFF_IRQ_ENABLE  0x1C
/* Register access macros */
#define NC_REG(base, off)       ((volatile uint32_t *)((uint8_t *)(base) + (off)))
#define NC_READ(base, off)      (*NC_REG(base, off))
#define NC_WRITE(base, off, v)  (*NC_REG(base, off) = (uint32_t)(v))
/* STATUS bits */
#define NC_STAT_SAMPLE_RDY  (1u << 0)  /* new sample available; write 1 to clear
*/
#define NC_STAT_BUSY        (1u << 1)  /* filter computing; read-only */
/* CONTROL bits */
#define NC_CTRL_BYPASS      (1u << 0)  /* 1 = pass mic through unfiltered */
#define NC_CTRL_RESET       (1u << 1)  /* write 1 to zero all coefficients */
/* Step size limits */
#define NC_STEP_MIN         0x0008u
#define NC_STEP_DEFAULT     0x0100u
#define NC_STEP_MAX         0x0800u
#define NC_STEP_INC         0x0080u    /* amount per key press */
/* Function declarations */
int      nc_init(volatile uint32_t **base_out);   /* open /dev/mem, map, set
defaults */
void     nc_close(volatile uint32_t *base);
int      nc_wait_sample(volatile uint32_t *base); /* poll STATUS[0]; clear on
return */
int16_t  nc_get_mic(volatile uint32_t *base);
int16_t  nc_get_ref(volatile uint32_t *base);
int16_t  nc_get_out(volatile uint32_t *base);
```

```
int      nc_toggle_bypass(volatile uint32_t *base);  /* returns new state */
void     nc_increase_step(volatile uint32_t *base);  /* clamps to NC_STEP_MAX */
void     nc_decrease_step(volatile uint32_t *base);  /* clamps to NC_STEP_MIN */
void     nc_reset_filter(volatile uint32_t *base);
#endif /* NOISE_CANCEL_H */
```