Technical Documentation
Noise Canceling Headphones
github.com/sssundar/NoiseCancelingHeadphones
E119C Spring & Summer 2015

Sushant Sundaresh
30 September 2015

# Table of Contents

Technical Documentation
Noise Canceling Headphones
github.com/sssundar/NoiseCancelingHeadphones
E119C Spring & Summer 2015

Sushant Sundaresh
30 September 2015

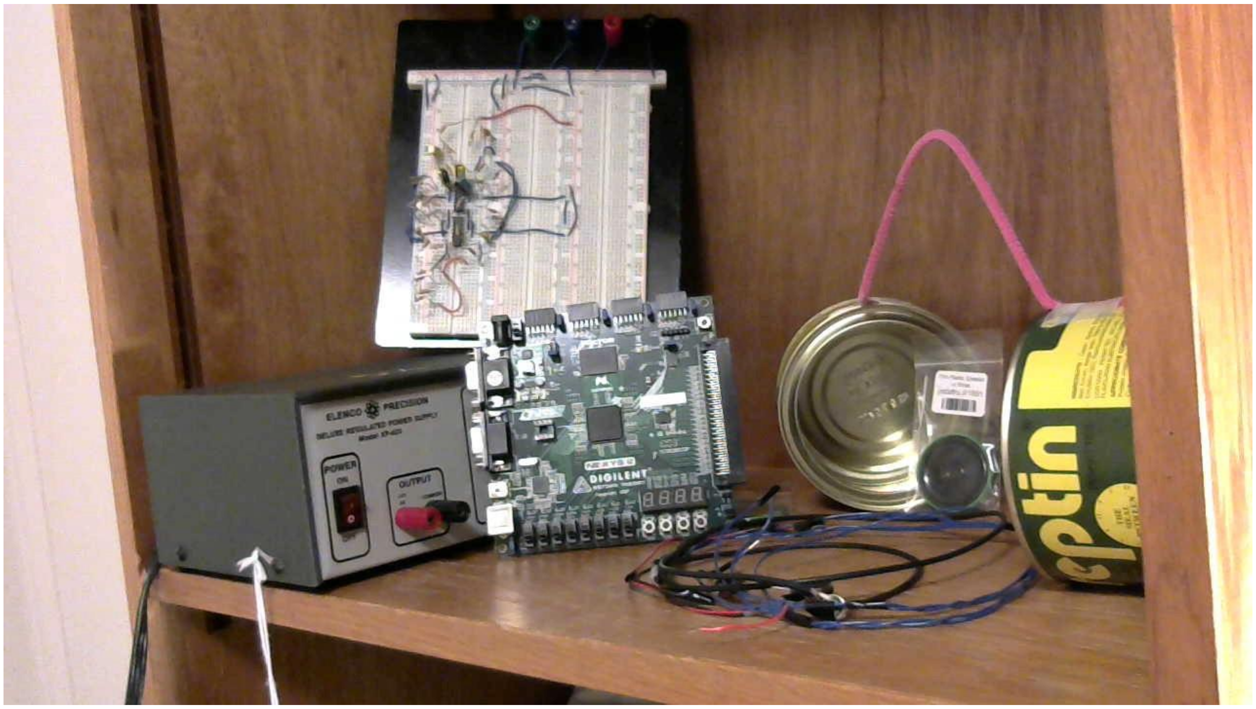
## System Description

The device is a pair of active noise canceling headphones cupping the ears.

There are two mics for each cup. The first is external, to sample input to be canceled. The second is internal, near the ear opening, to evaluate the error in cancelation.

The cups also have internal speakers which cancel sound near the ear when driven by an adaptive FIR filter acting on sampled input history. This filter is implemented within an FPGA.

Pictured below is a partial implementation to give a visual sense for the components involved.



## Block Diagrams & Analog Circuit Schematics

See pages 20-25; the system block diagram is on page 20.

## User Interface

A *User's Manual* is attached. See pages 26-27.

## Limitations

This design is a first-pass. It was intended to unearth 'unknown unknowns.' It is rough around the edges in terms of UI and applicability.

A significant amount of user interaction is required; the user must tell the filter when to start and stop adapting (it does not recognize when it has done well enough).

The amplitude of noise at the ear is expected to be around 1-3% of the input signal, even under optimal conditions, after about 40 seconds of adaptation, once the user has disabled further filter adaptation to allow the best filter to operate unperturbed. This error is a result of the restriction of the filter coefficients to powers-of-two, to make convolution simpler. In contrast, if a signed Q0.8 number system is used, restricting numbers in magnitude to between [0, 0.5], errors can drop into the 0.1% range within a much shorter time period, even if steps in the random walk are restricted to bit shifts.

The headphones will also likely produce low-amplitude high-frequency noise as a consequence of imperfect mechanical acoustic filter compensation and imperfect input audio prediction.

## Error Handling

If, on startup, the ADC10464 is triggered due to fluctuations on the PMOD outputs from the Nexys 2 dev-board, the ADC I/O controller designed for the FPGA can recover.

## Status

*Element*
Headphones

*Description*
Two metal enclosures with inner, outer mics, and speakers, and all wiring co-axial

*Status*
Partially Complete

*Block*
Mic Pre-Amp

*Description*
Low amplitude-distortion mic pre-amp daisy chain with a DC level shifter

*Status*
Implemented for left headphone cup

*Element*
Analog-to-Digital Converter

*Description*
Texas Instruments ADC10464 integrated circuit

*Status*
In Progress: soldered to breakout board; designed test VHDL. Implementing

*Block*
FPGA Clocks

*Description*
40 kHz sampling trigger
1.28 MHz sigma-delta DAC oversampling trigger
System clock as low-skew 2x divider off Nexys 2 dev-board 50MHz oscillator

*Status*
component designed and tested in VHDL
component designed and tested in VHDL
constraint map not yet attempted

*Block*
FPGA I/O Daisy-chain Synchronizer & Decoder

*Description*
Synchronizer for asynchronous ADC10464 interrupt, various buttons and switches

Combinational logic decoding: whether to allow filter updates, whether to allow sound output, whether to reset, which ADC samples drive the LED display

*Status*
component designed, partially implemented & tested

*Block*
FPGA ADC10464 I/O

*Description*
FSM to interface with ADC IC
Sample level shifting to kill DC component of ADC outputs, unsigned 8-bit integers
Output register to FPGA Left/Right Filter Pair block
Output register to LED array for mic pre-amp/level shifter calibration

*Status*
component designed & partially implemented, but not yet tested

*Block*
FPGA Left/Right Filter Pair

*Description*

### Common Elements

Max-length 64-bit LFSR Pseudo-Uniform Random Number Generator

Convolution Setup & Trigger FSM
Input History Updater FSM
Fractional Precision Shift-Multiplier FSM
Convolution Accumulator FSM

### Left/Right Filter Sub-Block

*Adaptation Sub-Sub-Block*
Error Residual Accumulator & Filter Updater FSM
Current Error Residual Magnitude Accumulator
Best Residual Register & Comparator
Best Filter Coefficient Array

*Working Memory Sub-Sub-Block*
Current Filter Coefficient Array Restricted to Signed Powers-of-Two
Input History Array

*Convolution Sub-Sub-Block*
Current Filter Coefficient Array Copy
Input History Array Copy
Accumulator with Fractional Resolution
Post-Accumulation Integer Truncator
Post-Truncation Level Shifter
8-bit Unsigned Over/Underflow Resolution
Output Register (to DAC)

*Status (for all)*
designed
tested in s/w
implemented in VHDL
tested in ISim
synthesized
fit in Spartan 3 (1200E) FPGA

*Block*

FPGA DAC Block

*Description*

Pair of sigma-delta 32x oversampled DACs with unsigned 8-bit inputs and 1-bit outputs

*Status*

specified via s/w simulation, designed, not yet implemented in h/w

*Block*

FPGA DAC Block Calibration

*Description*

Captures DAC and ADC blocks and outputs a sinusoid with peak-to-peak amplitude spanning the full 8-bit range. Displays the ADC samples from one internal mic on the LED array to allow the user to tune the DAC post-amp to match input-output scaling

*Status*

realized the need for this block after documenting summer work
not yet designed

*Block*

DAC Post-Amp

*Description*

Post-amplifier for the DAC output to limit current load on FPGA PMODs

*Status*

Designed, not yet tested

## Mic Pre-Amp Operation & ADC IC Configuration

On the attached schematics, please find the mic pre-amp circuit diagram on page 21.

The basic idea is as follows: the electret has an internal FET which, coupled with co-ax cables, helps preserve signal integrity through several feet of wire. At the breadboard level, this signal is immediately level shifted to kill the DC component at a time constant of 1 second to avoid distorting audio bands.

This AC signal is then amplified through a series of 4 daisy chained ~4x amplifier stages built using TL074CN's in non-inverting configurations. Op-amp power rails are grounded by a 10µF tantalum cap close to the pin.

The end result is a 256-fold amplified signal of amplitude roughly 0.5V at reasonable noise levels. This signal is level-shifted up by a variable pot and must be calibrated to match the internal '0' level of the ADC as described in the *User's Manual.*

The ADC IC has not been tested yet, but I intend to test it first with a 2.5V-3.3V power supply, off the FPGA to match logic levels between the two. In the worst case I will use a 5V input and a 3.3V-to-5V bidirectional logic converter, then configure the Pmod IOs as LVTTL.

A 2.5V Vcc for the ADC means, roughly, that if we take the 8 LSB off the data bus (to maximize resolution), we need to ensure that '0' + 0.5V stays below that cutoff for the 2 MSB. Moreover the level must be set such that the ADC samples roughly 128 when the mic is disconnected.

## FPGA I/O Daisy-chain Synchronizer & Decoder Operation

See the attached schematic for this block on page 22.

A daisy chain synchronizer was demonstrated for EE119B's GCD demo, and I found it worked well in tests writing to registers based on button pushes. The chain is simply a 3-mer of DFFs, the last of which is deemed synchronized.

The decoder logic is simple enough: most signals go straight through. SW6,7 are exceptions. These are *sound enable/filter adaptation enable*, respectively. When either is low, filter adaptation is not allowed, to avoid updating the filter in open loop. When *filter adaptation enable* alone is low, sound is enabled, allowing the user to hear the result of the best filter, instead of the potential cacophony of a filter which sometimes audibly walks in the wrong direction before the update logic fixes the issue.

## FPGA Filter Block Timing & Sizing: Estimates vs. Results

Simulation in the design stages of this project determined that 40kHz sampling was sufficient to detect and cancel periodic noise in the audible spectrum.  DAC 32x oversampling required a 1.28 MHz DAC integrator trigger at least.

Back-of-the-envelope timing analysis of the filter, ADC, and DAC design suggested that the critical path lay in the convolution accumulator, requiring the latching of the largest ripple carry adder chain in the system in a single clock cycle.

Therefore, so long as synthesis projected a max clock frequency greater than 25MHz, I figured it would be possible to divide the 50MHz crystal oscillator down to 20MHz using on-chip rational division and distribution functionality. This 20MHz base clock could be further divided down by 512 or 16 to yield a 39kHz or 1.25 MHz trigger, exactly as required.

All estimates that follow are for a 32-coefficient filter restricted in magnitude from [1/128, 1/2], with 8-bit inputs extended by 3 bits precision for convolution.

As a sanity check, notice that the longest FSM in the filter block (and the entire system) takes roughly 376 clocks to operate (see the test bench for convolution). This means sample processing has terminated *well* before the 40kHz trigger (512 clocks) to process the next sample.

In addition, since the ADC IC is rated for 1MHz conversions, and certainly for 200kHz, we know 4 conversions from the 4 mics can easily happen in the interval between 40kHz triggers.

Synthesis of the filter block (as a test, since the full system is not complete yet) suggested a ridiculous 140MHz max speed for a Spartan 3-1200E with a -4 speed grade. I suspect some error but still feel a 20MHz 16-bit ripple carry adder is quite reasonable (the max length in the convolution accumulator).

So, the clocking strategy outlined above is expected to work.

As a sanity check on sizing, I predicted close to 3000 DFFs and triple that in combinational logic, and found roughly 3300 flip-flops (19%), 3000 slices (43%) and 7000 4-input LUTs (39%) utilized for the test synthesis of the filter block as spec'd. I knew sizing would not be an issue given that the entire architecture is based on shifts.

## FPGA Clocks Operation

The two 40kHz and 1.25MHz triggers are, as described above, merely 512x and 16x divisions off a 20MHz base clock. These being powers of two, triggering is as simple as looking for MSB carry out off the counter incrementor.

I am still working out the details of how to divide the 50 MHz oscillator by 2/5 to get to 20 MHz. The reference manual for the Nexys 2 makes it crystal clear this is possible.

# FPGA ADC10464 I/O Block Operation

The ADC block performs a synchronization handshake with the peripheral ADC10464 IC to latch, every 40kHz sampling cycle, all 4 of the mic inputs.

Please see the associated block diagram on page 20. The ADC10464 breakout board pin settings are described on page 21.

This block outputs

- 3 active low control signals – chip-select (CS_), read (RD_), and sample (S_), to Pmods
- 2 selector bits (S0, S1) to choose which of the 4 mics the ADC should digitize, to Pmods
- 4 signed 8-bit output registers with the unsigned 8-bit ADC samples shifted down by 128
- 1x 8-bit LED calibration output register (see *User's Manual, Calibration)*

It takes as input

- a synchronized active low interrupt (INT_) from the ADC IC, signaling digitization
- 4 synchronized button inputs to choose which sample to display on LEDs for calibration
- 8-bits direct from the Pmods linked to the data bus of the ADC IC; we synchronize these via handshaking

It operates as follows:

- CS_ is low always
- the FSM waits with RD_ low & S_ high, clearing the sample counter (S0, S1)
- the FSM sees a 40kHz sampling trigger and waits one cycle with RD_ high
    - on full reset, i.e. every system startup if the user follows instructions, this ensures that the ADC IC, if accidentally triggered, has ample time to convert, signal completion, and receive this mock "acknowledge" that we've read the output. This ensures the IC is idle before we request any more processing
- the FSM sets S_ low, then high, then waits for INT_ low. It then acknowledges with RD_ low, waits for INT_ to return high
- It then copies the data from the ADC data bus to its internal registers. The MSB is complemented to level shift by the 'zero' value the mic analog level shifters are calibrated to match
- it increments the sample counter & cycles until all samples have been latched, then idles

# FPGA Left/Right Filter Pair Operation

This block is the workhorse of the project. The algorithm it implements was fleshed out using the script *system_restricted_to_powers_of_two.m*. After highlighting key design decisions, we'll dive into the operation.

1. The Left/Right filters update independently. Coefficients are represented as a sign bit and a positive 8-bit power of two, and walk only along this number line when asked to 'update':
   $+2^{-1} \ldots +2^{-8}, 0, -2^{-8} \ldots -2^{-1}$

2. Working filter coefficients are asked to randomly 'update' using a pseudo-RNG once every 2K samples, where K is the length of the input history in samples.

3. Error signals from internal mics are accumulated as well, to track the effectiveness of the working filter coefficients. The ADC block outputs error signals without a DC component, if calibrated properly. The absolute value of these signals is my chosen error residual.

4. In the *User's Manual,* the user is asked to reset the system with filter adaptation *off,* partially to allow the LFSR to mix itself up from the reset state before we use it as a pseudo-RNG

5. When convolving, input history is multiplied by filter coefficients using a shift-multiplier; 3 fractional precision bits are preserved and accumulated.

6. The convolution accumulator has been sized to integrate fractional precision values as well, and is only truncated to integer precision after convolution is complete. This is critical.

7. The filter, for now, must be told via user input to adapt or stop adapting. This is because, in simulation, two-tone systems took up to 40 seconds to cancel. I wanted the flexibility to choose the adaptation time on the fly for the first-pass & testing.

8. Every FSM and any register that isn't explicitly cleared every cycle has a global reset.

Let us consider a single filter out of the pair required for independent left/right headphone adaptation. The relevant schematics are on pages 23-25. It will be easiest to explain filter operation by example:

- A 40kHz convolution trigger signals.
- The convolution setup controller copies the latest external mic input sample over to the filter block, updating the input history
- In parallel the filter update controller copies the latest internal mic error signal over to the residual accumulator
- The setup controller clears the convolution input history
    - It then copies the working filter and input history to the convolution block
    - It triggers a shift-update of the input history to prepare for the next sample, and triggers the convolution multiplier
- The filter update block in parallel has accumulated the magnitude of the error signal
    - By this point the working filter has been copied and is free to be mutated.
    - The block checks whether or not the filter needs to be updated: is this the 2Kth sample in this update interval?
    - If so, it compares the working and best residuals
        - It updates the best filter with the working filter if appropriate.
        - It copies the new best filter coefficients to the working coefficients
        - It wipes the working error
        - It clears its sample counter.
        - It then asks the working filter to randomly update using the LFSR
    - If not, the filter update block updates its sample counter and idles
- The convolution block in parallel has been merrily multiplying coefficients with inputs
    - Coefficients are either 0 or a positive power of two with a sign bit on the side
    - So, it shifts its coefficients left until a 1 shifts out, and shifts its inputs right at the same time
    - It preserves 3 extra bits of fractional precision for products
    - If an input has a coefficient of zero, it clears itself when asked to shift-multiply
    - Once the multiplication is done, the sign bit of the coefficient determines whether or not the input needs to negate itself
    - Finally the convolution accumulator is triggered
- The convolution accumulator shifts in product terms one at a time
    - It sign extends and integrates them. It retains fractional precision
    - When it accumulates all K products, it truncates the result to an integer
    - In s/w simulation it was found that truncating this integer down to 8 bits did not adversely affect adaptation
    - Therefore this block level shifts its result by the same amount we removed in the ADC block, then maps negative results to 0, results over 8 bits to 255, and anything in between is output directly to the DAC
- All controllers are now idle and the filter block waits for the next convolution trigger

# FPGA DAC Block Operation

The associated diagrams for this block are on pages 21-22.

The DAC block has two internal sigma-delta converters, which each take as input

- an 8-bit unsigned integer
- a ≈40kHz trigger to update the current sample being processed
- a ≈1.28MHz oversampling trigger
- an active high sound output-enable flag

It outputs a 1-bit stream that may be passed directly to a Pmod and post-amp.

The post amp is both a power amp and a variable attenuator, to make sure [0,255] at the DAC block input translates to a volume equivalent to [0,255] at the external mic inputs, allowing filter coefficients to remain small. For now, this attenuator must currently be calibrated manually, by trial and error.

I recognized the necessity for a dedicated calibration block for the DAC only when writing up this review of my summer progress, and have added this block requirement to the *Status* listing.

The post-amp contains:

- a sub-unity gain op-amp in an inverting configuration in series with
- a DC-zero (timescale 1 second to avoid audible poles), in series with
- an inverting buffer, in series with
- a (default) 20x gain power amp, the LM386N-1, the output of which is directly passed to a 0.25W, 8Ω nominal speaker.

The sigma-delta conversion is straightforward. No FSM is required. If sound output is disabled the block operates normally but the output to the Pmod is gated low.

- Initially on every sample update the integrator is cleared and the quantizer output is set to 0.
- A new 8-bit unsigned sample is loaded from the filter block output register. The quantizer output is extended to 8-bits (0 = 0, 1 = 255). This unsigned result is subtracted from the sample without mutating the sample. The result cannot underflow and automatically converts to an 8 bit signed integer.
- This value is integrated (after sign extending 3 bits) by an 11 bit signed accumulator with two extra bits that in s/w has been confirmed not to over or underflow. The integrator updates on the oversampling trigger (32x the rate of the sample trigger).
- The integrator state is quantized as negative or positive (0, 1) and this result feeds back to the combinational quantizer-extender and also is output to a Pmod.

*Aside:* The script *wav2List.py* helped me 'hear' the effect of variations on bit resolution and over sampling rate for this block. The script *dac.py* helped spec the integrator oversampling rate (32x), and the integrator overflow-protection bit-width (+2 bits overflow, +1 bit sign).

# Inputs

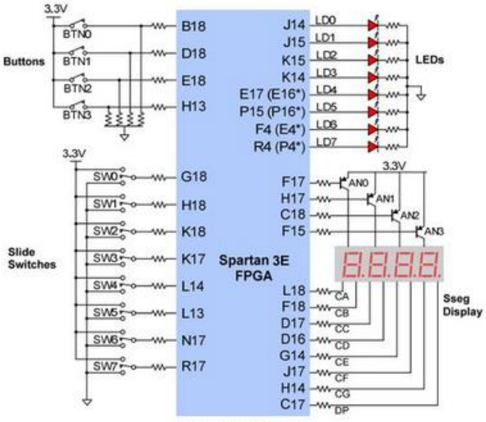
The system samples an analog input at ≈40kHz from two electret mics per headphone cup. One is external, facing outward near the center of the outer surface of the cup. The other is internal, near the ear opening.

These mics have an approximate passband of 400 Hz – 20 kHz with a known gentle resonance around 10 kHz. Mic signals are amplified roughly 4^4 times in a 4x TL074CN amplifier daisy chain up with low gain distortion but significant phase distortion, up to an amplitude of 0.5V at comfortable ambient sound levels.

These 4 signals are then passed into an ADC10464 IC to be digitized at roughly 200kHz. The net ≈40kHz sampling has non-trivial but, for a first pass, acceptable microsecond jitter in phase delays between samples. The dev board PMOD inputs accepting the ADC data bus are shown on the following page.

The system also takes the following switch & button input from the dev board, mapped to the FPGA pins as shown. All are interpreted as active high.

| *Input ID* | *Name* | *Description* |
|---:|---:|---:|
| SW5 | RESET | Full reset of core registers and FSMs |
| SW7 | Allow Filter Adaptation | Adapt filter to new sounds<br>May sound horrible while adapting<br>Otherwise, use the best filter found so far |
| SW6 | Allow Sound Output | Drive speakers, or not, to hear the difference |
| Power Switch | Dev Board Power | Start board, load configuration from ROM |
| BTN 0 | Show Left External Mic In | Show ADC samples from this input on LEDs |
| BTN 1 | Show Left Internal Mic In | Show ADC samples from this input on LEDs |
| BTN 2 | Show Right External Mic In | Show ADC samples from this input on LEDs |
| BTN 3 | Show Right Internal Mic In | Show ADC samples from this input on LEDs |



* pin numbers for -1200 die

Reference: http://www.instructables.com/id/Nexys-2-Whack-A-Mole/step12/Connecting-the-Inputs-and-Outputs/

# Outputs

Normally, the only output from the system sensed by the user is audio from the speakers embedded in the headphone cans.

This audio is generated off an FIR filter convolution with sample input history from the external mics. Left and right sample histories are convolved with independent filters.

The result of this filtering is fed into a (≈1.28MHz 32x oversampled) sigma-delta DAC, attenuated ~20x, then buffered by a 20x-minimum gain LM386N-1 power amp before being fed into each speaker.

The distortion accrued is acceptable for a first-pass system dealing only in adaptation to periodic signals. This is because the feedback signal to the adaptation filter, the error at the internal mic, must go near 0 for a decent filter. Low amplitudes are low amplitudes regardless of phase distortion.

The system can also display ADC samples on the dev board LED array as a calibration output. See *User's Manual* for details. The FPGA pin map is shown in the figure on the previous page.

| *Input ID* | *Name* | *Description* |
|---:|---:|---:|
| LD0-7 | See Mic DC Level Calibration | An 8-bit array of LEDs for input level display |

Finally, the system outputs multiple control signals to the ADC, and the DAC output as well. These are linked to PMOD outputs as shown below.

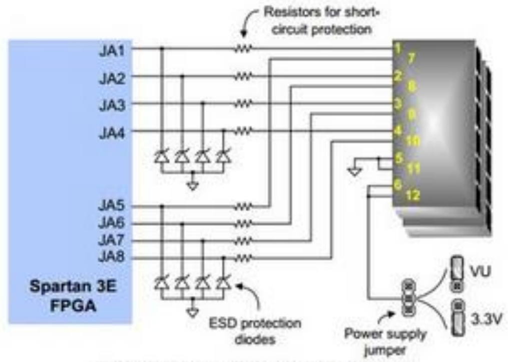


Figure 23: Nexys2 Pmod connector circuits

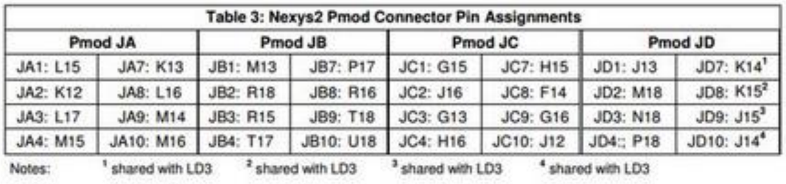


Table 3: Nexys2 Pmod Connector Pin Assignments

| Pmod JA | | Pmod JB | | Pmod JC | | Pmod JD | |
|---|---|---|---|---|---|---|---|
| JA1: L15 | JA7: K13 | JB1: M13 | JB7: P17 | JC1: G15 | JC7: H15 | JD1: J13 | JD7: K14[1] |
| JA2: K12 | JA8: L16 | JB2: R18 | JB8: R16 | JC2: J16 | JC8: F14 | JD2: M18 | JD8: K15[2] |
| JA3: L17 | JA9: M14 | JB3: R15 | JB9: T18 | JC3: G13 | JC9: G16 | JD3: N18 | JD9: J15[3] |
| JA4: M15 | JA10: M16 | JB4: T17 | JB10: U18 | JC4: H16 | JC10: J12 | JD4:; P18 | JD10: J14[4] |

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3

Reference: http://www.instructables.com/id/Nexys-2-Whack-A-Mole/step12/Connecting-the-Inputs-and-Outputs/

# Part Listing

## headphones

1. Cui CMA-4544PF-W omnidirectional electret microphone
    a. 20-20kHz with clearly spec'd frequency response
2. Adafruit PID 1891, Thin Plastic Speakers, 8 Ω, 0.25 W
3. 2x Threptin biscuit cans

## mic pre-amp & ADC

4. TL074CN op-amp
5. 10-bit ADC 600ns, 28-SOIC, ADC10464CIWM/NOPB-ND IC
6. SMT Breakout PCB for SOIC-28

## dev board

7. Nexys 2 with Spartan3-1200E FPGA

## speaker power amp

8. LM386N-1 power amplifier

## power supply

9. ±[1.5,15] V, +5V (3A) Elenco Precision XP-620

## miscellaneous

10. Circuit Elements
    a. 0.1-10µF tantalum/ceramic/electrolytic caps
    b. 100-100kΩ resistor set
    c. co-axial/solid-core wiring
    d. large solderless breadboard
    e. male header pieces for breakouts
    f. LEDs, push-buttons
11. Soldering Aids
    a. MG Chemicals No-Clean Flux Paste (Cat. no. 8341)
    b. Lighted Magnifying Glass & Solder Vacuum
12. Measurement Aids
    a. Mastech MS8261 Digital Multimeter
13. Wiring Aids
    a. Klein wire strippers
    b. electrical tape & static discharge grounding bracelet
14. Crafts Tools
    a. X-ACTO #2 blade
    b. pipe-cleaners, rubber bands

# Code Listing

| | |
|---|---|
| • System Structure | *Description* |
| o system_wiring.vhd | Overall Structure |
| • Constants | |
| o magic_numbers.vhd | FSM States and Bitwidths |
| • Commonly Used Elements | |
| o kBitCounter.vhd | a k-bit counter |
| o nCounter.vhd | a counter to n |
| o AddSub.vhd | a full adder |
| • Pseudo-Random Number Generator | |
| o lfsr_maximal_xnor_64_bit.vhd | a 64 bit max-len LFSR |
| • Filter Block | |
| o Structure | |
| ▪ block_filter_pair.vhd | Filter Block structure |
| ▪ block_filter.vhd | Single Filter Structure |
| o Common Finite State Machines | |
| ▪ fsm_load_next_samples.vhd | setup for convolution |
| ▪ fsm_input_history_shift.vhd | prep for ADC samples |
| ▪ fsm_convolution.vhd | convolve |
| o Convolution Block | |
| ▪ accumulator_convolution.vhd | sum up inputs x coefficients |
| ▪ register_convolution_filter_coefficient.vhd | helps implement shift-mul |
| ▪ register_convolution_input.vhd | helps implement shift-mul |
| o Filter Update Block | |
| ▪ fsm_update_filter.vhd | update filter based on error |
| ▪ Input History | |
| • register_input_history.vhd | store input history for filter |
| ▪ Working Filter | |
| • register_filter_coefficient.vhd | uses LFSR to walk in $\{0,\pm 2^{-n}\}$ |
| ▪ Best Filter | |
| • register_best_filter_coefficient.vhd | the best filter seen so far |
| • register_best_residual.vhd | its associated error |
| • ADC Block | |
| o led_counter.vhd | an unfinished ADC I/O FSM |
| o led_counter.ucf | port mapping play |
| o README | says “not done yet” |
| • DAC Block | |
| o n/a | not implemented yet |

# Test Listing

Please see the files below for documentation on testing.

*Simulations Verifying Implementation*

The filename describes the entity or block tested.

- test_block_filter.vhd
- test_fsm_update_filter.vhd
- test_lfsr_xnor_64.vhd
- test_convolution.vhd
- test_accumulator_convolution.vhd
- test_register_convolution_filter_coefficient.vhd
- test_register_convolution_input.vhd
- test_register_filter_coefficient.vhd
- test_input_history_shift.vhd

| *Simulations Guiding Design* | *Description* |
|---|---|
| • DAC Block | |
| o Python | |
| ▪ dac.py | exploring DAC |
| ▪ wav2list.py | audio writer to hear results |
| • Filter Block | |
| o Matlab | |
| ▪ system_restricted_to_powers_of_two.m | exploring filter adaptation |

Technical Documentation
Noise Canceling Headphones
github.com/sssundar/NoiseCancelingHeadphones
E119C Spring & Summer 2015

Sushant Sundaresh
30 September 2015

## Evolution of Goals & Approach

I've been working on this project, so far, from April-September. I estimate roughly 3 months of solid work have gone into this so far.

This being my first major design effort, I thought, for when I pick things up again, it'd be useful to document the evolution of my approach.

My high-level design targets were

1. the headphones themselves
2. an analog sensor interface to the headphone mics
3. a DAC with a tiny I/O footprint to the headphone speakers
4. a s/w algorithm for processing sensor input
5. a h/w implementation of this algorithm

I expected that targets 3 & 4 would stretch my analytical abilities and have many 'unknown unknowns.' I expected targets 2 & 5 to be substantial design challenges but 'known unknowns,' as Glen had thrown such designs at us many times before.

### *April*

April was spent deciding on a plan of attack for Target 4. That is, testing ideas using actual speakers/mics interfaced with Matlab Simulink, and through simulation.

These efforts made it clear within three weeks that the scope of my project needed to scale down from the hilariously optimistic "cancel human voices in real time" to "cancel periodic sounds within a few ms using feedback."

I chose analog hardware, built various versions of target #2 as the plan for target #4 fleshed out, making the requirements on the analog interface (tolerance for distortion) more clear.

I played with a Raspberry Pi 2 for two weeks before deciding it would not make a good s/w dev platform for target #4.

I did three revisions of a design for target #1, prototyping along the way. This solidified my familiarity with the mics/speakers and their various resonances.

### *May*

Come May I felt I had a good handle on targets #1, 2 & 4 (as it turned out, mistakenly) and shifted my attention to target #3, understanding the sigma-delta DAC algorithm, which promised low-audible noise DAC output via a single I/O port per speaker. This was valuable to me because at the time I wanted to limit the PMOD I/O on my Nexys 2 dev board to avoid having to interface with the high-speed expansion bus.

s/w simulation of the sigma-delta algorithm helped me spec out the numbers on oversampling requirements, integrator bit-widths, minimum clock speeds for 40kHz base audio sampling, and so on.

*June-August*

Come June I realized that all my tests had been done on single-tone systems. Adding a second tone caused my LMS-feedback structure to become unstable.

From mid-June to August's end I spent most of my time interning. I worked a total of 2 weeks on the headphones. I tried several variations on the feedback structure, testing how these responded to noise, other tones, etc. Nothing I tried solved the stability issue.

*September*

In September, I decided my actions had spoken. I had failed to analytically identify the source of the instability, and I had not been able to find a solution through trial and error.

So, I made things simple.

From this point on, I made decisions as follows: I simply wanted to know if a decent corrective filter existed in a physical implementation of my design. I wanted to hear to believe. I wanted a prototype, not just s/w exploration.

I removed the constraints of

- time (it no longer needed to be ms-fast, it could take up to a minute)
- generality (it would be tested for multi-tone periodic signals only)
- accuracy (up to 3% error in noise-canceling at the ear would be acceptable)

I chose to implement a pseudo-random walk in FIR filter space, rather than using LMS to update coefficients. I updated every once in a while based on the running residual noise level measured at the ear.

While s/w testing this, I found the FIR filter could be further restricted to

+2^-1 ... +2^-8 0 -2^-8 ... -2^-1

so long as the samples were 8-bit ADC inputs, meaning the filter could be easily enlarged to 32 coefficients on my dev board, as shifter/accumulator based convolution is much simpler than MACs.

Now, I started to see rapid progress. The s/w algorithm was fleshed out, then all arithmetic was made binary, and bounds were placed on how long the thing typically needed to run to cancel simple two-tone inputs. I sized how large coefficients would be, how precise all arithmetic would be, and then over the course of a week drew up the h/w designs for the filter block.

I spent the next week implementing this filter block in VHDL, testing it incrementally.

Finally, in the last days of September I started interfacing with the ADC chip using the FPGA, testing synchronization and working out port-maps and IO pin constraints to avoid inadvertently triggering the ADC chip on power-on.

Based on previous work for target #3, the DAC algorithm, I finished the h/w design for the DAC.

On September 30, 2015, I was 100% confident that I had the know-how required to complete the project. I estimated I was 24 solid hours of work away from a complete, working prototype – roughly two days.

However, I consciously stopped building and started documenting. I did not start the implementation of target #3 in VHDL, and I did not finish that of target #2 either.

Good thing, too! I only recognized the necessity for a dedicated calibration block for the DAC when writing up this review of my summer progress.

During the fall term I intend to focus on new course projects, the SAE 2016 competition, and my job search. Headphones, I'll see you again during winter break!

## Process Observations

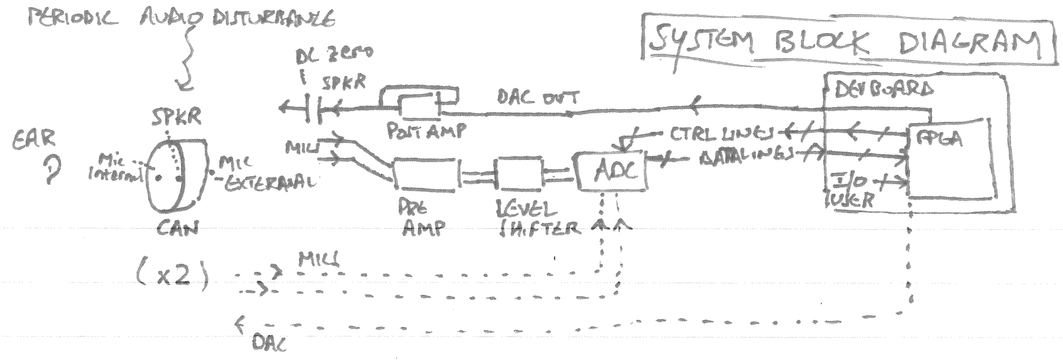
- Plan out goals, designs, and tests explicitly
- Document both process and engineering decisions
- A diary guides thought and makes asking for help much easier
- Take unfamiliar problems one step at a time with hard deadlines for action
- Iteration over perfection

## Acknowledgements

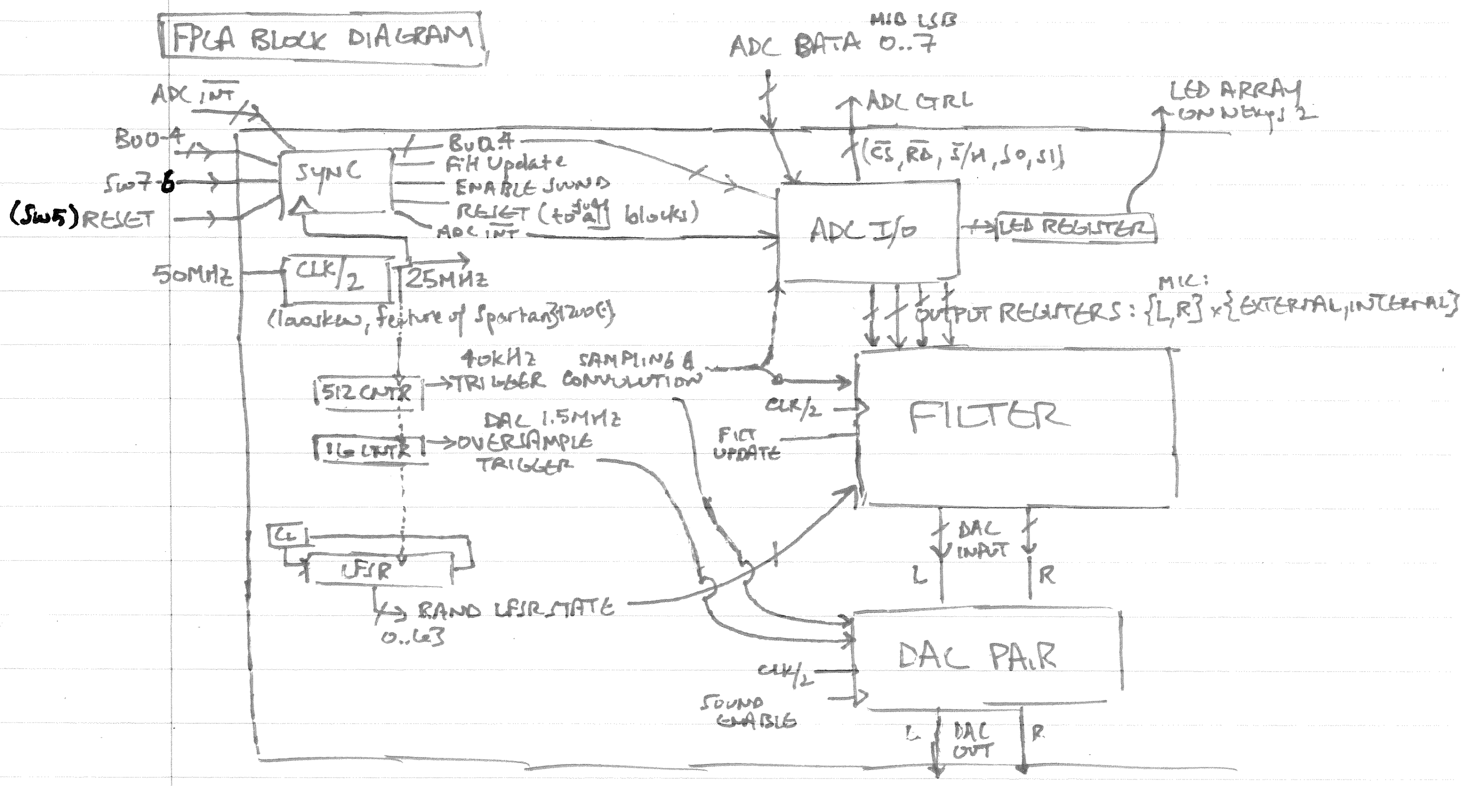
to my family
to Glen George
to Max Wang

Thank you

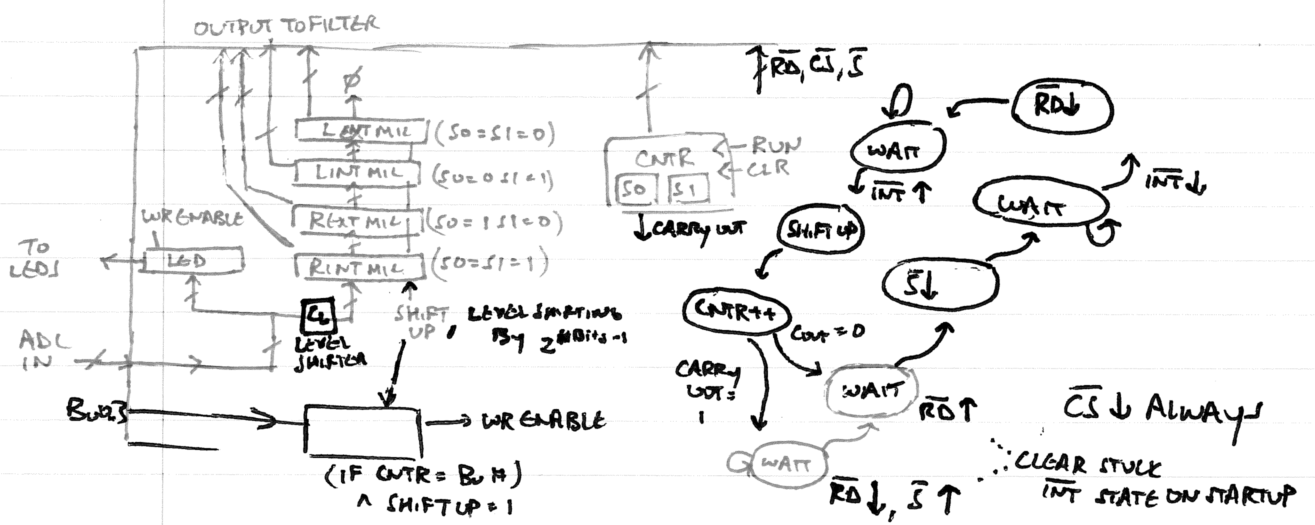
## SYSTEM BLOCK DIAGRAM



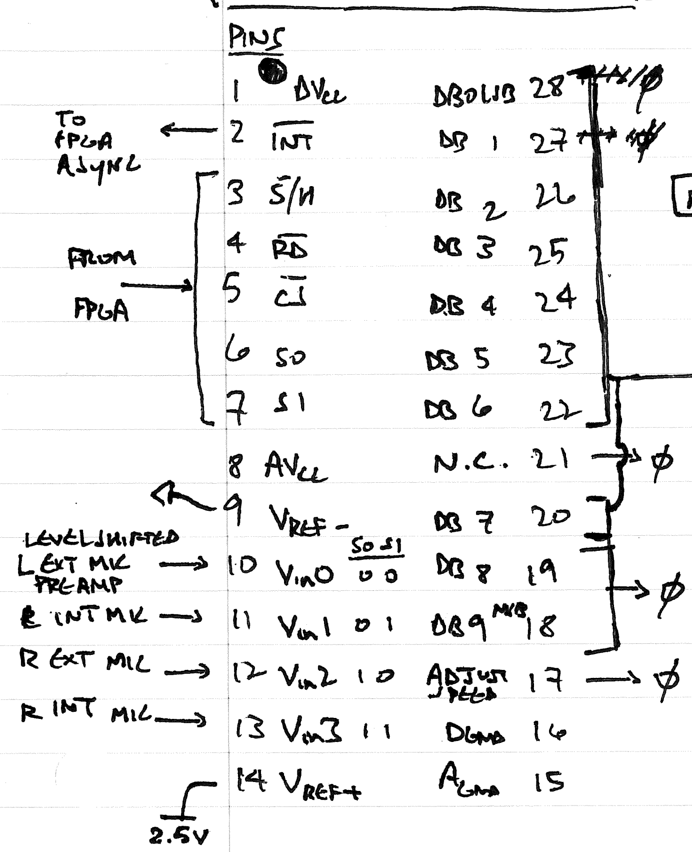
## FPGA BLOCK DIAGRAM



## ADC I/O BLOCK

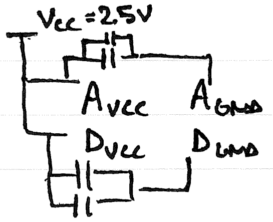


## ADC10464 Breadboard Setup



PINS

| | | | | |
|---|---|---|---|---|
| 1 | $DV_{cc}$ | | DB0 LSB | 28 |
| 2 | $\overline{INT}$ | | DB 1 | 27 |
| 3 | $S/\overline{H}$ | | DB 2 | 26 |
| 4 | $\overline{RD}$ | | DB 3 | 25 |
| 5 | $\overline{CS}$ | | DB 4 | 24 |
| 6 | S0 | | DB 5 | 23 |
| 7 | S1 | | DB 6 | 22 |
| 8 | $AV_{cc}$ | | N.C. | 21 |
| 9 | $V_{REF-}$ | S0 S1 | DB 7 | 20 |
| 10 | $V_{in}0$ | 0 0 | DB 8 | 19 |
| 11 | $V_{in}1$ | 0 1 | DB9 MSB | 18 |
| 12 | $V_{in}2$ | 1 0 | ADJUST SPEED | 17 |
| 13 | $V_{in}3$ | 1 1 | $D_{GND}$ | 16 |
| 14 | $V_{REF+}$ | | $A_{GND}$ | 15 |



CERAMIC 0.1μF || 10μF TANTALUM

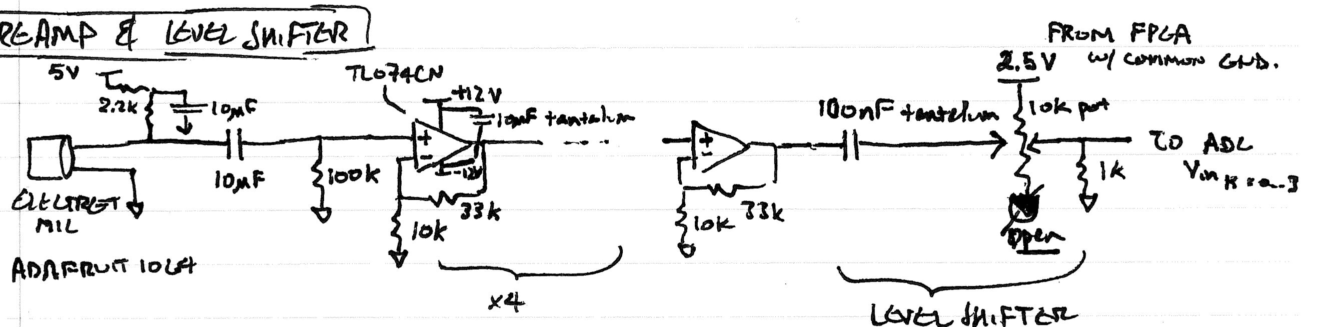
### My INTENT:

1) $V_{cc} = 2.5$ ON ATTEMPT 1 TO MATCH LOGIC LEVELS OFF LVCMOS25 PMODS ON NEXYS2.

2) AUDIO OFF PREAMP ~ 400mV in AMPLITUDE SO WOULD EASILY FIT IN [0,2.5]v if level shifted.

3) SINCE RESTRICTING TO 8 ~~L~~MSB (for resolution), NEED TO TUNE LEVEL SHIFT AFTER PREAMP TO AVOID CLIPPING.

## MIC PREAMP & LEVEL SHIFTER

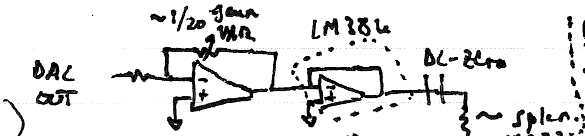


## DAC POSTAMP THOUGHTS

LM386 supports 325mW. My spkrs are 0.25W; 8Ω nominal. w/ 20x gain default.

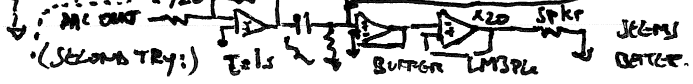
A pre-attenuator to scale the DAC output (0-2.5V) followed by a LM386 20x gain → a DC-zeroing cap → spkr seems a good approach.

So: (FIRST ATTEMPT)



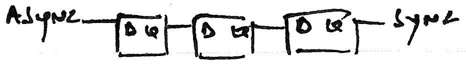
BUT, A REASONABLE BLOCKING CAP w/ NOMINAL 8Ω SPEAKERS WOULD INTRODUCE POLES INTO THE AUDIBLE SPECTRUM, SO,

(SECOND TRY:)



SEEMS BETTER.

## SYNC BLOCK



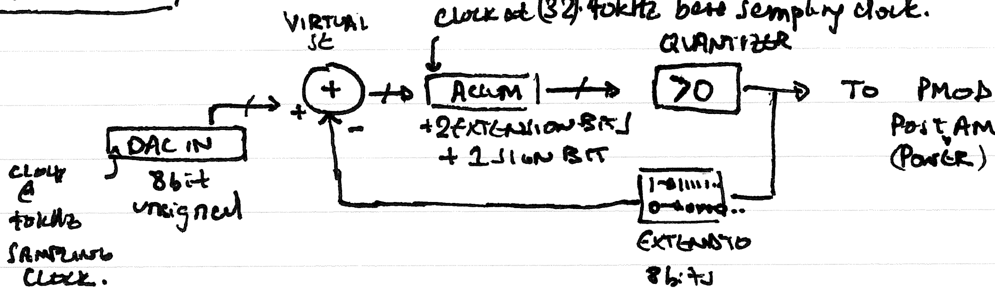
: SYNCHRONIZER FOR ALL ASYNC SIGNALS:

- BU 0..3
- SW 7..6
- RESET (SW5)
- $\overline{\text{INT}}$ (ADC)

DECODE LOGIC
- SW 7 (FILTER ADAPTATION ENABLE)
- SW 6 (SOUND ENABLE) ⟶ DISABLING AUTO-DISABLES FILTER UPDATE (do not want to update when not closed loop).
- SW 5 (RESET)

⟶ ∴ SW 6 OR SW7=0 ⟹ FILT UPDATE NOT ALLOWED

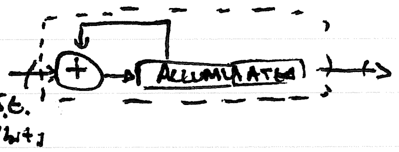
## DAC BLOCK



By VIRTUAL SE, I mean, treat $\overset{+}{\underset{-}{\oplus}}$ ⟶ AS A SIGNED 8 bit #. NO OVERFLOW IS POSSIBLE.

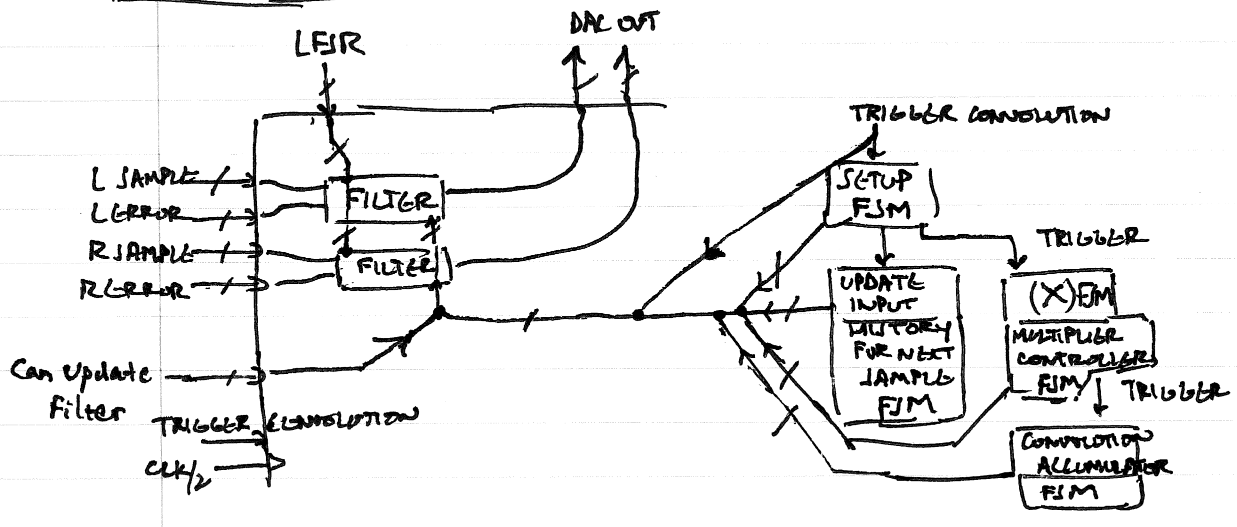
So if MSB of ⊕ is set, sign extend 3 bits to match integrator width.

ACCUM is an integrator:

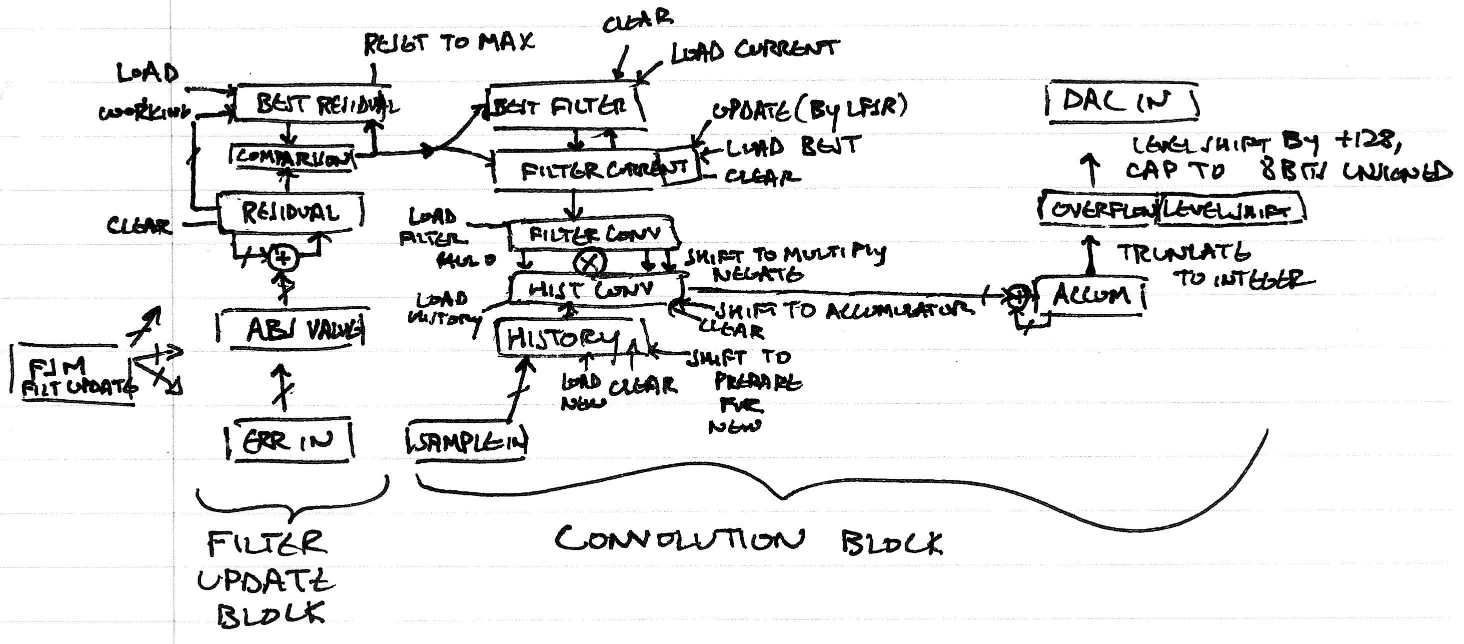


We know from synthesizing the Filter this 8+3=11 bit adder is not in the critical path.

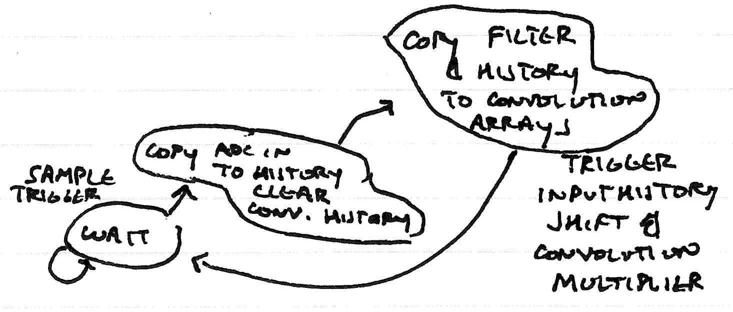
## FILTER BLOCK



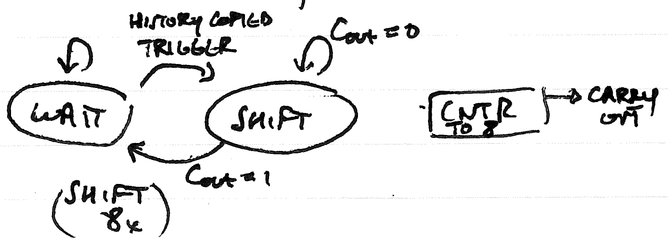
## FILTER



## FSM SETUP & TRIGGER CONVOLUTION

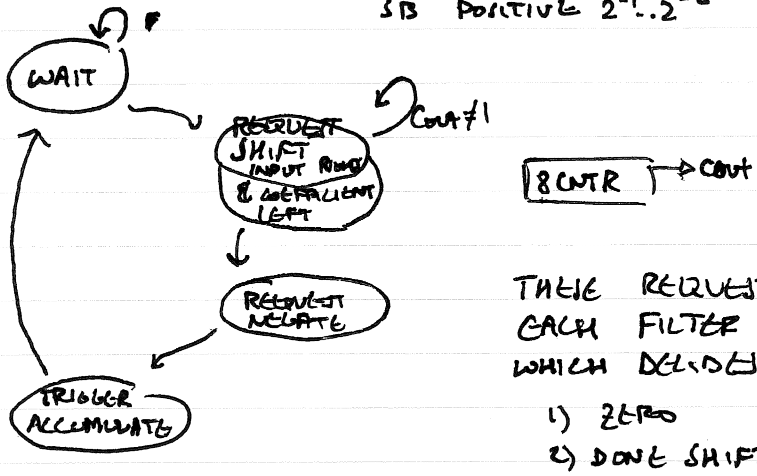

COPY FILTER & HISTORY TO CONVOLUTION ARRAYS
SAMPLE TRIGGER
COPY ADC IN TO HISTORY CLEAR CONV. HISTORY
WAIT
TRIGGER INPUT HISTORY SHIFT & CONVOLUTION MULTIPLIER

## FSM UPDATE INPUT HISTORY



## FSM MULTIPLIER

RECALL COEFFICIENTS WALK IN $+2^{-1} .. +2^{-8}$ 0 $-2^{-8} .. -2^{-1}$
SO a · b is just shift operations & negation,
as coefficients are stored as $\underbrace{\ }_{SB}$ $\underbrace{\qquad\qquad}_{\text{POSITIVE } 2^{-1}..2^{-8}}$

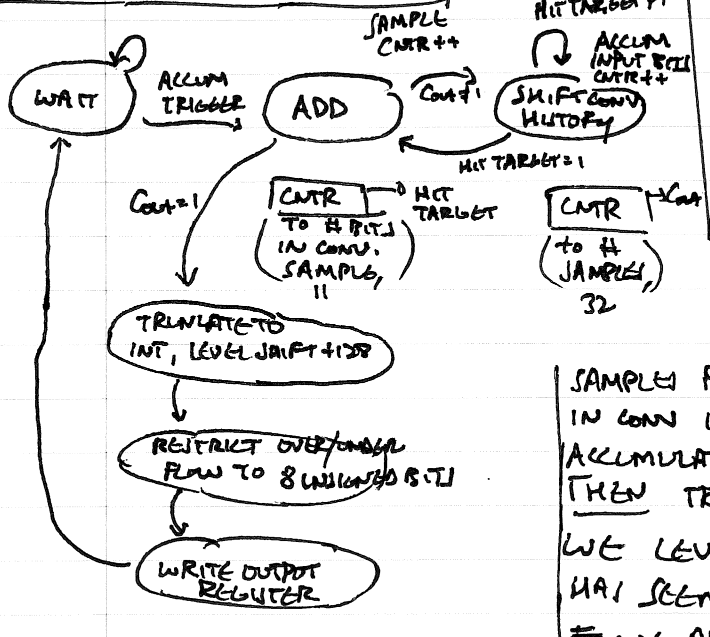


THESE REQUESTS GO TO EACH FILTER COEFFICIENT WHICH DECIDES IF IT IS

1) ZERO
2) DONE SHIFTING (NOT ZERO BUT JUST SHIFTED A 1 OUT OF MSB)
3) NEGATIVE

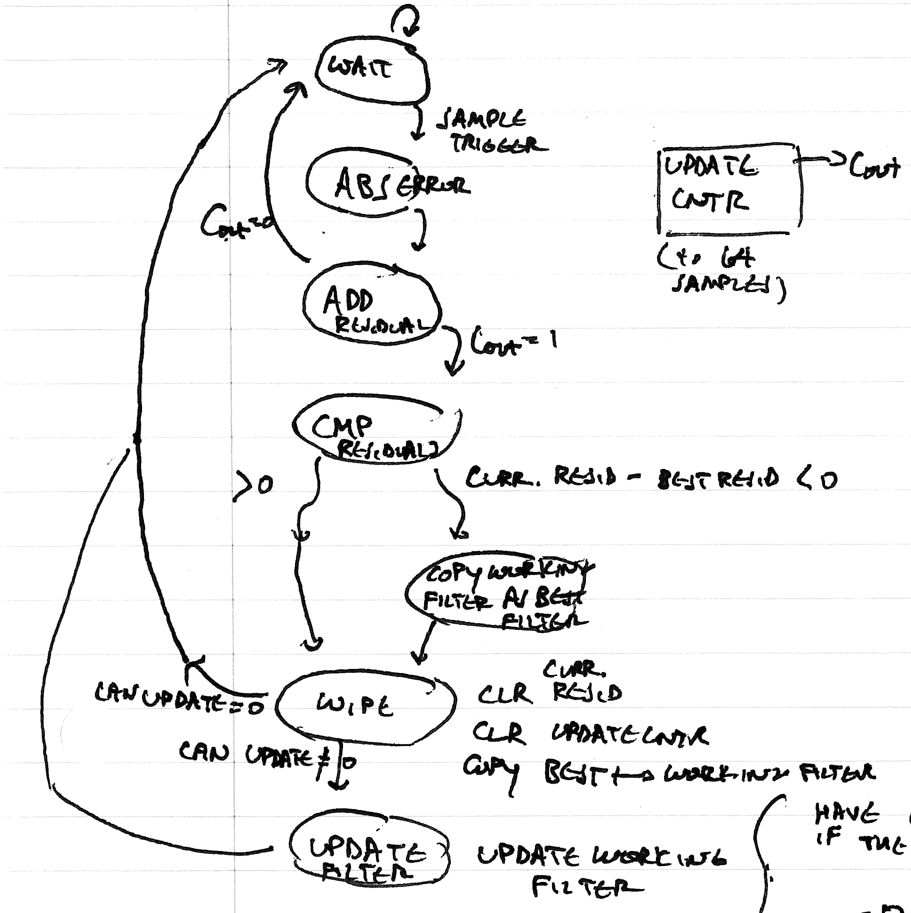
IF NEGATIVE, NEGATE REQUESTS GO THROUGH TO "CONVOLUTION INPUT"
IF (2) SHIFT REQUESTS DO NOT.
IF (1) SHIFT REQ. DO PASS & THE CONV. INPUT CLEARS ITSELF AS IT SEES ITS COEFF IS ZERO.

## FSM CONV. ACCUMULATOR



SAMPLES FROM ADC - 8 bits
IN CONV INPUT, EXTENDED TO 1/2, 1/4, 1/8 PRECISION
ACCUMULATOR RETAINS THIS WHILE ADDING
THEN TRUNCATES JUST BEFORE OUTPUT.
WE LEVEL SHIFT AS THE FILTER TILL NOW
HAS SEEN 0-DC component (LEVEL SHIFTED INPUT)
WE ADD IT BACK TO MAKE DAC LOGIC SIMPLER.

# FSM UPDATE FILTER



HAVE 64 LFSR BITS & 32 COEFFICIENTS
IF THE 2 BITS SENT TO A COEFF ARE:

N/A

- 0, 1 DO NOTHING
- 2 SHIFT ←
- 3 SHIFT →

WHERE → ← ARE ON <u>THIS</u> NUMBERLINE

$$+2^{-1} \cdots +2^{-8} \quad 0 \quad -2^{-8} \cdots -2^{-1}$$

← →

User's Manual
Noise Canceling Headphones
github.com/sssundar/NoiseCancelingHeadphones
E119C Spring 2015

Sushant Sundaresh
30 September 2015

## Description

This pair of adaptive noise-canceling headphones cancels periodic sounds in the vicinity of its inner microphones. The headphones should have two mics (external, internal) and one speaker on each cup. They achieve roughly 30x noise amplitude reduction after 1 minute of adaptation.

## Usage

- These headphones actively attempt to cancel periodic audible sounds
- To operate, connect via USB for power and flip the power switch on the Nexys 2 dev board
- Then plug in the breadboard power supply (preset to output 5 V)
- Set the *Allow Filter Adaptation* switch to *off*
- *Set* the *Allow Sound Output* switch to *on*
- Toggle the *RESET* switch from *off* to *on* to *off*
- Place the headphone cups over your ears
- Set the *Allow Filter Adaptation* s witch to *on*
- Wait 40 seconds, then toggle *Allow Filter Adaptation* to *off*
- The periodic noise amplitude should have been reduced by roughly 30x

## Calibration

There are three ways the device might become uncalibrated.

First, the breadboard power supply might be tuned to something other than 5 V.

Second, all mic pre-amps pass their signals through a DC level-shifter before ADC sampling. The FPGA removes this DC component before processing, and if this DC level is made too high or too low on the breadboard (by fiddling with the potentiometer) there is a chance the filtering algorithm might fail. The buttons labeled 0-4 on the dev board correspond to 'external left mic,' 'internal left mic,' 'external right mic,' and 'internal right mic' respectively. Pressing these buttons will display the 8-bit data being sampled by the ADC, raw, on the dev board LEDs, with the LED labeled LD0 as the most-significant bit. Press only one of these buttons at a time, and tune the pre-amp pots in a quiet environment until this MSB is almost always on, and most other LEDs are fairly dim. This means your mic inputs have a reasonably calibrated DC component.

Third, the post-amps to the headphone speakers from the dev board scaled to match input sound volume to output sound volume. If this post-amp gain is misadjusted, you have no recourse, now, but to fiddle with it until the headphones seem to output sound at roughly the same level at whatever noise they are trying to cancel. A DAC calibration module will be added in a future revision.

## Limitations

If the sound is too loud, the analog interface amplifying mic inputs can clip. It is still being tested whether loud aperiodic sounds during an attempted cancelation can cause adaptation to fail.

## User I/O

| *Input ID* | *Name* | *Description* |
|---:|---:|---:|
| SW5 | RESET | Full reset of device |
| SW7 | Allow Filter Adaptation | Adapt filter to new sounds<br>May sound horrible while adapting<br>Otherwise, use the best filter found so far |
| SW6 | Allow Sound Output | Drive speakers, or not, to hear the difference |
| Power Switch | Dev Board Power | Start board, load configuration from ROM |
| BTN 0 | Show Left External Mic In | Show ADC samples from this input on LEDs |
| BTN 1 | Show Left Internal Mic In | Show ADC samples from this input on LEDs |
| BTN 2 | Show Right External Mic In | Show ADC samples from this input on LEDs |
| BTN 3 | Show Right Internal Mic In | Show ADC samples from this input on LEDs |
| LD0-7 | See Mic DC Level Calibration | An 8-bit array of LEDs for input level display |

## Operation

1. During adaptation to a new sound, on each headphone cup, an external mic is sampled as an input to be canceled, and an internal mic is sampled as an error signal, a measure of how low the noise level is, at the ear.

2. These inputs are routed through coaxial cables to a pre-amp stage, then level shifted and sampled at 40kHz, with unsigned 8-bit resolution, using an ADC IC.

3. The FPGA level-shifts these inputs back to remove their DC component (calibrated). It regularly checks the error to decide whether the working cancelation filter is better than its previous best; it also regularly updates its working filter in a random walk.

4. The FPGA filters the input history it has collected (FIR filtering) against its working cancelation filter, and outputs this convolution result, with 8 bit resolution, to a sigma-delta DAC, also implemented on the FPGA.

5. The DAC outputs a single binary stream oversampled 32x at roughly 1.28MHz. This stream, when low-pass-filtered by a speaker, replicates the convolution result. The stream is actually fed into a post-amp that drives the headphone speaker.