University College of Southeast Norway

Real-time audio signal processing for UAV Based Applications Using the Xilinx Zynq SoC

Master thesis in Systems Engineering with Embedded Systems

Zhili Shao

Supervised by: Dr. António L. L. Ramos and Dr. José A. Apolinário Jr.

June 9, 2017

Faculty of Technology and Maritime Sciences

Outline

1. Introduction
2. Implementation
3. Experiments
4. Conclusion and Future work

Introduction

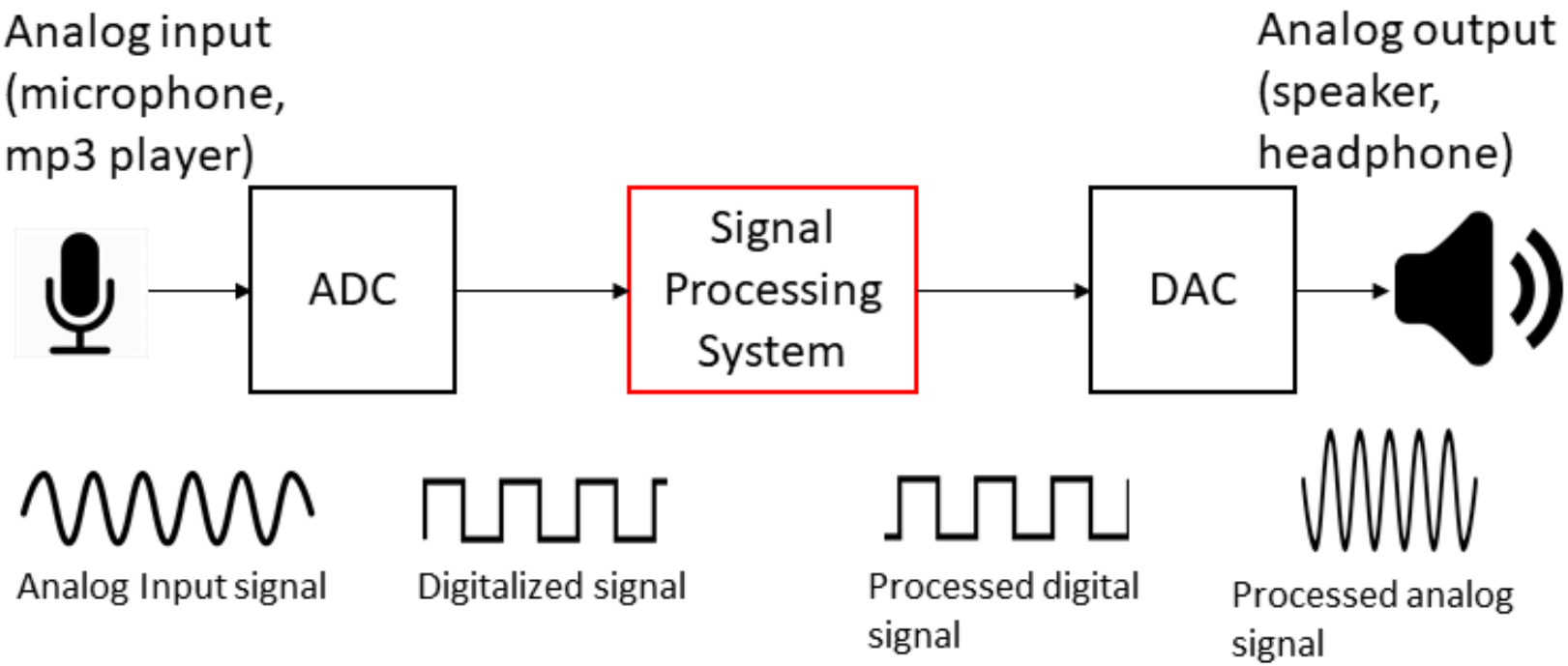
UAV applications

- Military:
  - weapon carrier
  - battlefield surveillance
  - network relay
  - ...
- Civil:
  - aerial photography
  - disaster rescue
  - sensor network data collection
  - cellular network reinforce
  - precision agriculture
  - ...

However, there is almost no audio application on UAVs. Why?

- noise
- working condition
- no perceived needs

Audio Digital Signal Processing



Digital signal processing (DSP) manipulates digitized signals with the purpose of filtering, measuring, compressing and sound reproducing.

Digital Filters

A discrete-time system using a sequence $x(k)$ signals as input, has an output $y(k)$ through a transformation expressed as in

$$y(k) = T\{x(K)\}, \tag{1}$$

$T\{\}$ is the system function.

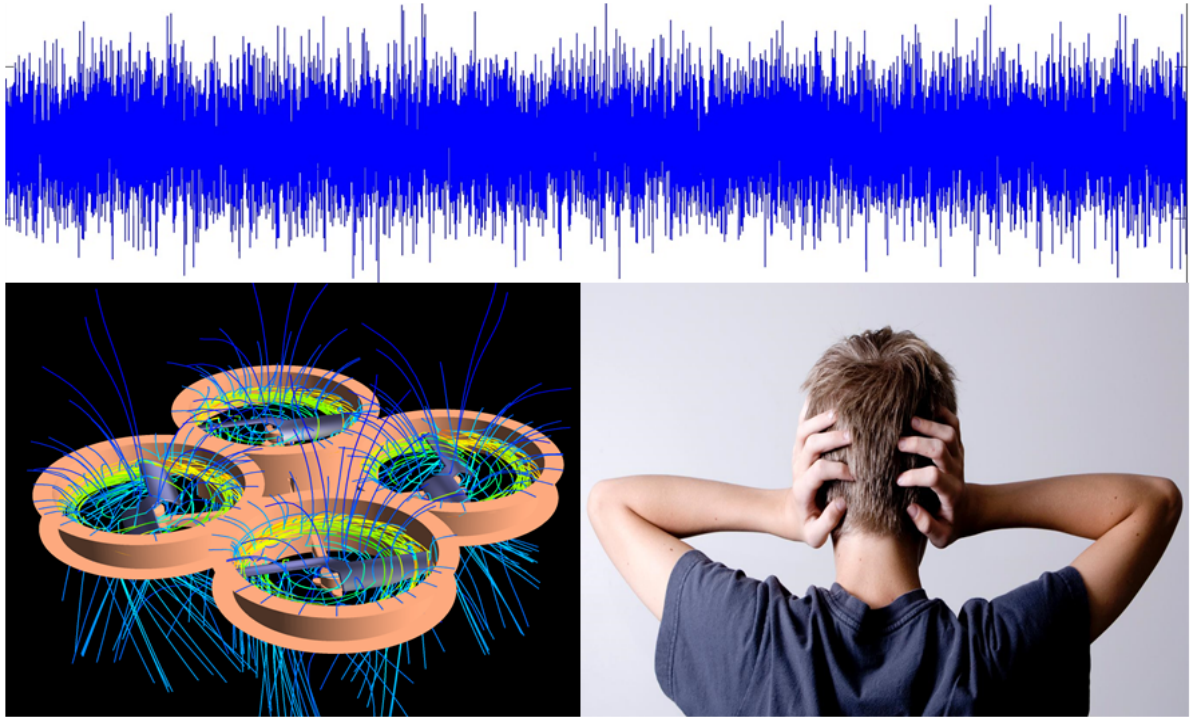
Digital filter applications

- FIR filter: low-pass filter, band-pass filter, high-pass filter.
- Adaptive filter: system identification, noise cancellation.
- Median filter: removing salt and pepper noise in imaging signal processing.

Possible audio application on UAVs

Solution	Application
Adaptive filter	Propeller noise reduction
FIR band-pass filter	Speech enhancement
Median filter	Gunshot enhancement

Propeller noise reduction



Objective: Decrease effects of propeller noise.

Solution: adaptive filter.

Speech enhancement

Objective: Extracting specific frequency domain.

Solution: FIR band-pass filter

Gunshot enhancement

Objective: Gunshot signal enhancement before destination of arrival estimation.

Solution: median filter.

Objective

Design a real-time audio signal processing platform for UAV based applications.

Methodologies and Technologies

- Hardware and software codesign.

Methodologies and Technologies

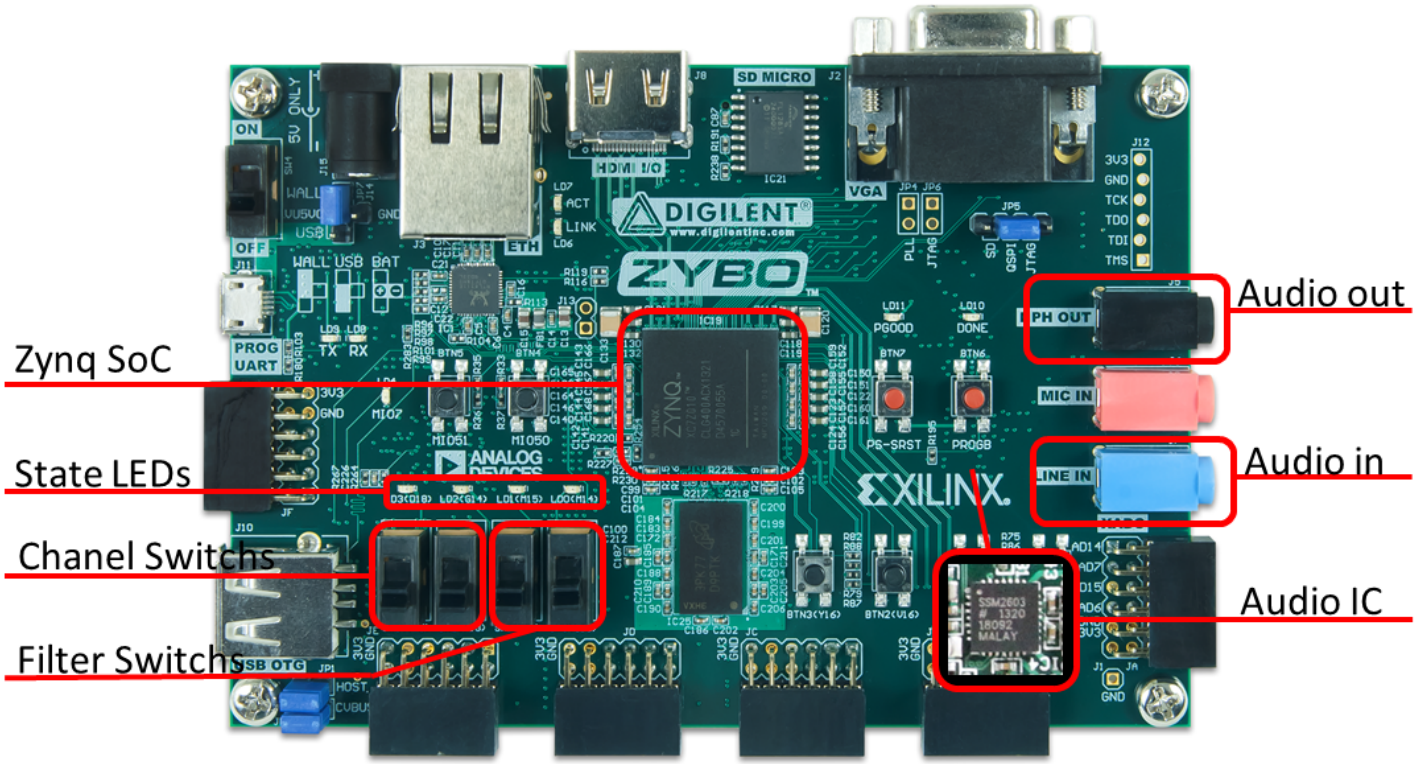
- Hardware and software codesign.
- Zynq SoC from Xilinx.

Methodologies and Technologies

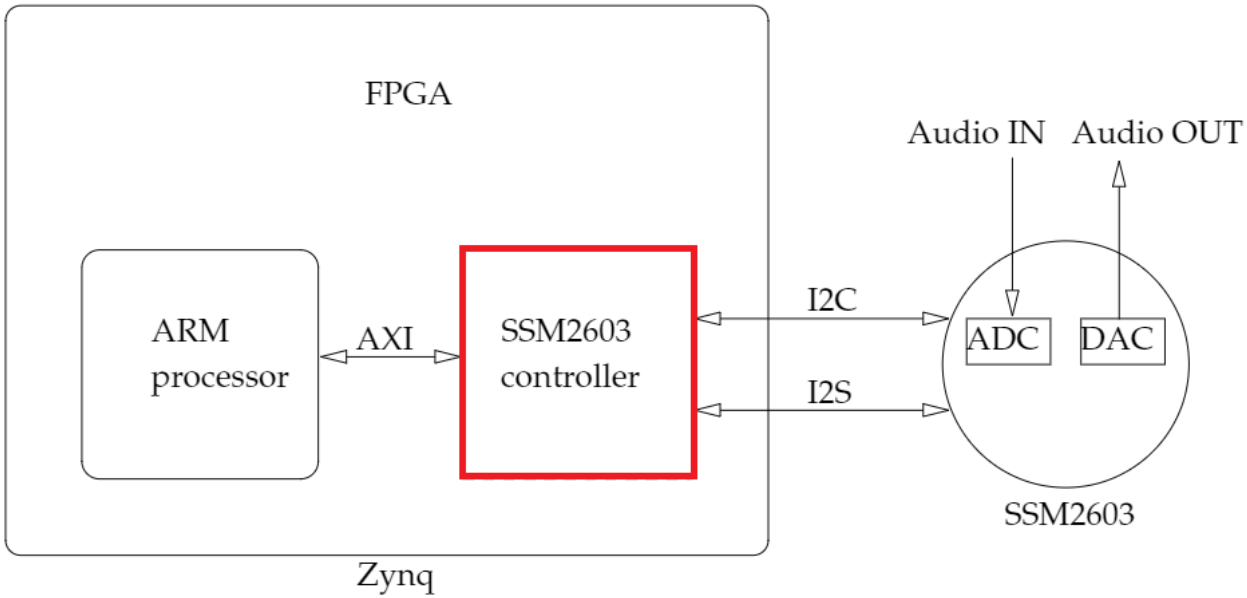
- Hardware and software codesign.
- Zynq SoC from Xilinx.
- Inter-chip communication protocols: AXI, I2C, I2S.

Implementation

Development Environment

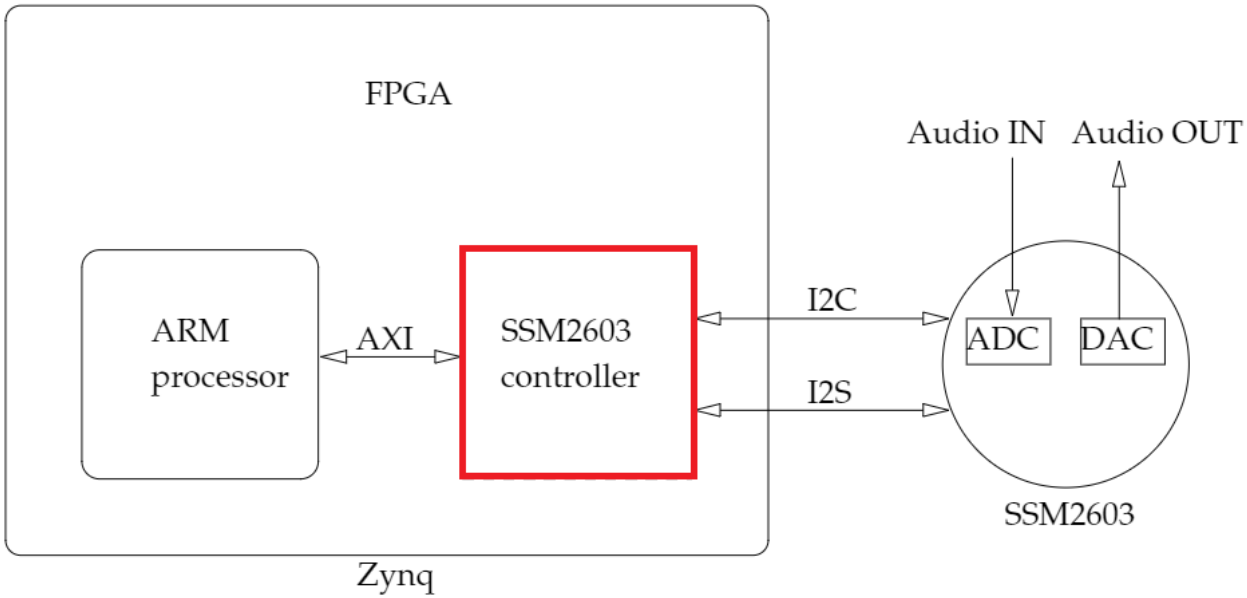


DSP hardware structure



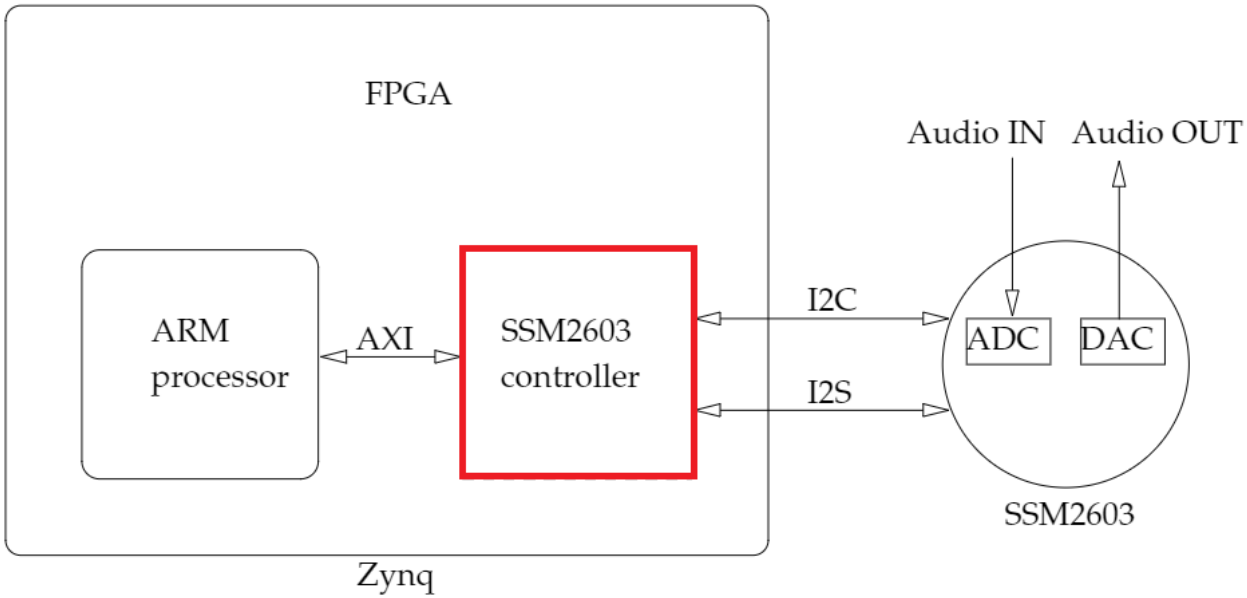
- **Processor:** It is a dual-core ARM Cortex-A9 processor, one part of Xilinx Zynq-7000, AP SoC architecture.

DSP hardware structure



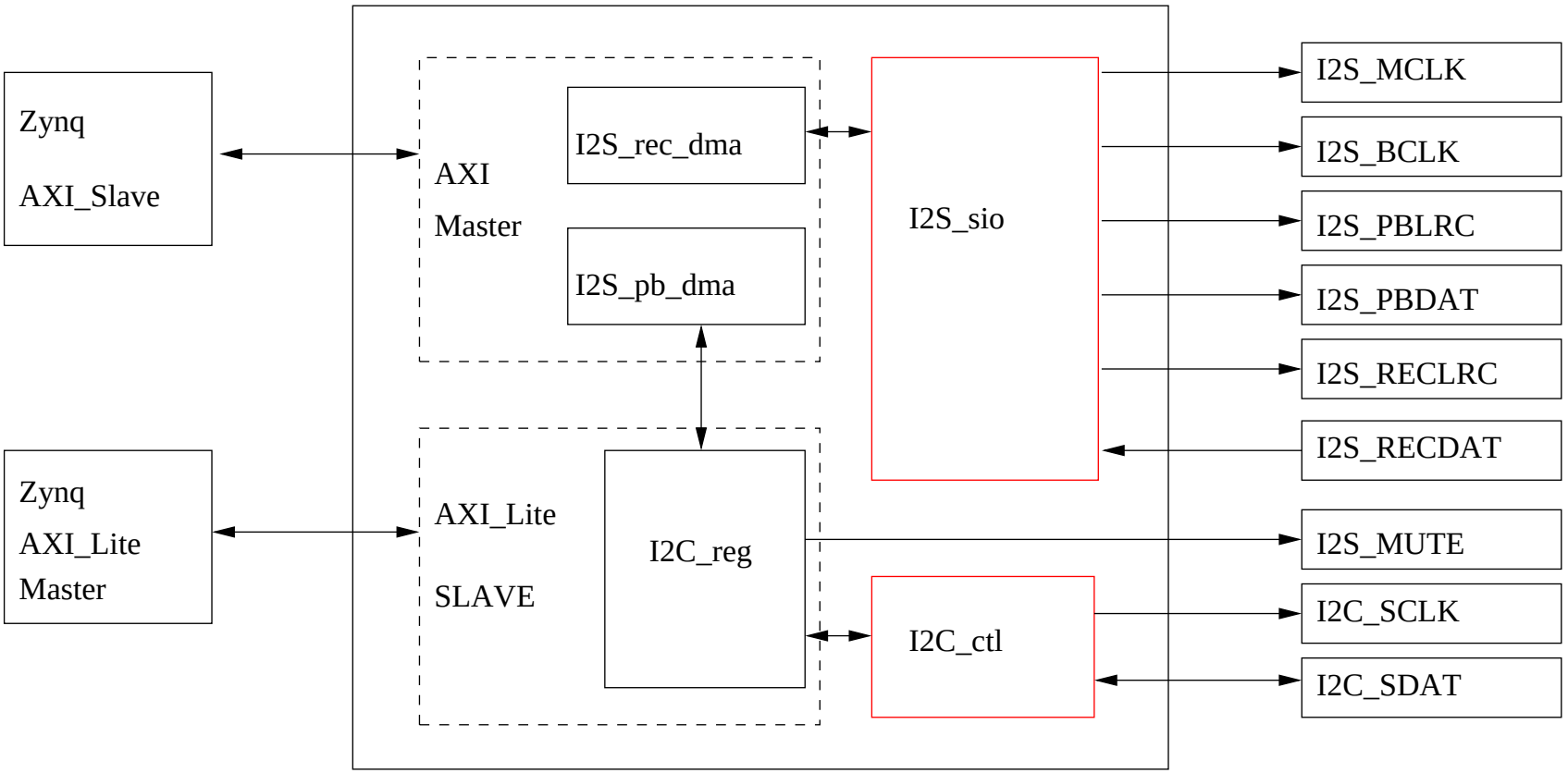
- **Processor:** It is a dual-core ARM Cortex-A9 processor, one part of Xilinx Zynq-7000, AP SoC architecture.
- **Audio Codec IC:** SSM2603 from Analog Devices, providing stereo, 48 kHz sampling rate, 16-bit ADC and DAC conversion in this case.

DSP hardware structure



- **Processor:** It is a dual-core ARM Cortex-A9 processor, one part of Xilinx Zynq-7000, AP SoC architecture.
- **Audio Codec IC:** SSM2603 from Analog Devices, providing stereo, 48 kHz sampling rate, 16-bit ADC and DAC conversion in this case.
- **Codec IC controller:** It is implemented on FPGA part of the Zynq SoC, specially designed for I2C and I2S protocols communication between ARM processor and SSM2603.

SSM2603 controller IP



Software design

Filter implement:

- FIR filter
- NLMS adaptive filter
- Median filter

Common techniques:

- Circular buffer

Software design

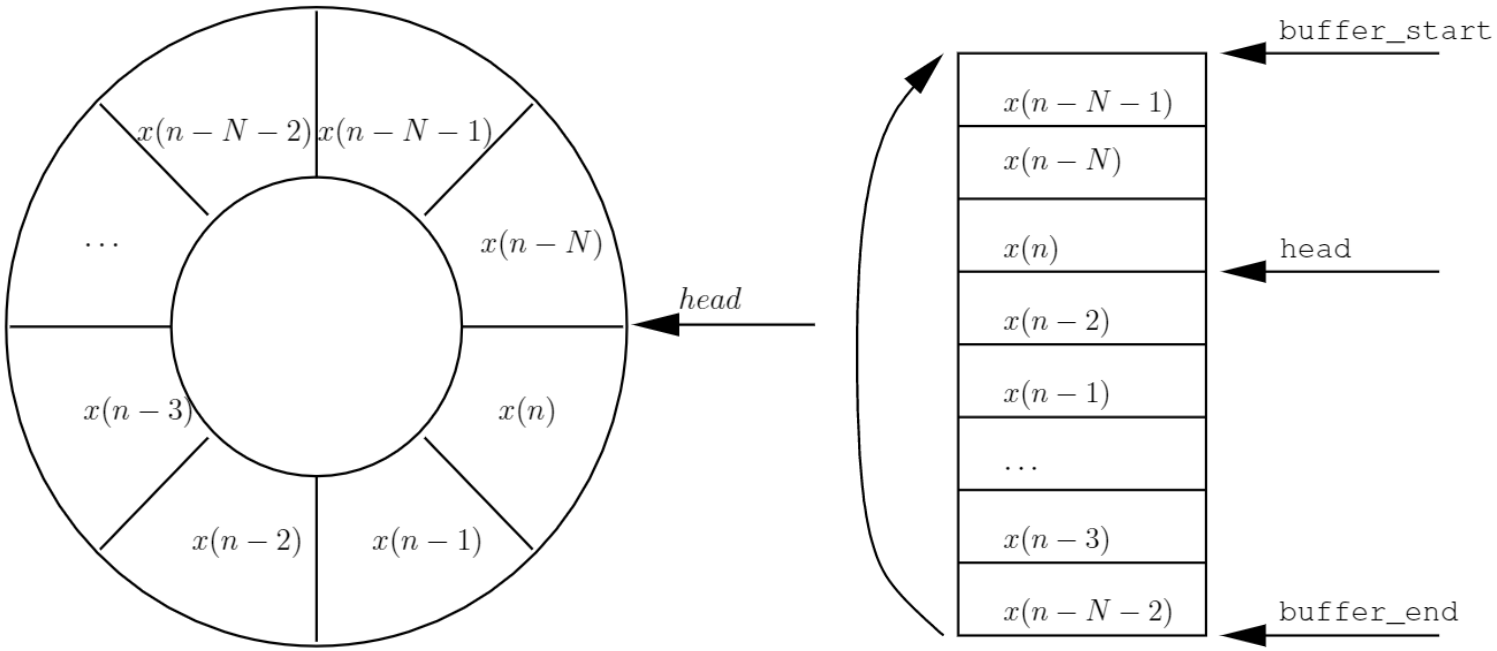
Filter implement:

- FIR filter
- NLMS adaptive filter
- Median filter

Common techniques:

- Circular buffer
- Fixed-point arithmetic

Circular buffer



Fixed-point calculation

Table 1: Fixed-point floating-point comparison

Fixed-point	Floating-point
Limited dynamic range	Large dynsmic range
Low power consumption	high power consumption
No need For FPU, suit for FPGA and processor without FPU	need FPU support
Overflow and quantization error	easy to program since no scaling is required

Q format

Q format for a signed, binary scaled fixed point number:

$$QM.N \tag{2}$$

- Q stands for the sign.
- M is the bit amount to the left of the imaginary decimal.
- N is the bit quantity to the right of the imaginary decimal.

$Q7.8$ can represent a range from -128.996 to $+127.996$, its precision is $1/2^8 = 1/256$.

Fixed-point Multiplication

- $QM.N * QM.N = Q(2M+1).(2N)$.
- The result is been scaled by extra N, right shift N to go back $QM.N$ format.
- the intermediate result need more bits:$16bits * 16bits = 32bits$.

FIR filter

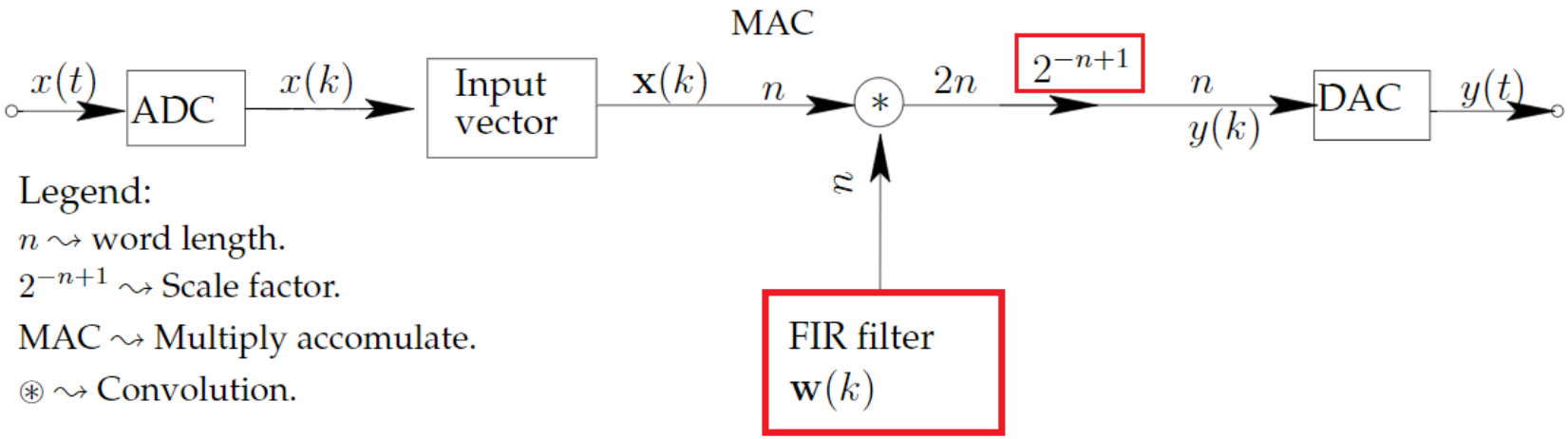
If the discrete-time FIR filter has order N. The output $y(k)$ is the weighted sum of input $x(n)$ with filter coefficients array:

$$[w_0 \quad w_1 \quad \cdots \quad w_N], \tag{3}$$

as shown in

$$y(k) = \sum_{i=0}^{M} w_i x(k-i). \tag{4}$$

Implement of FIR filter



Legend:

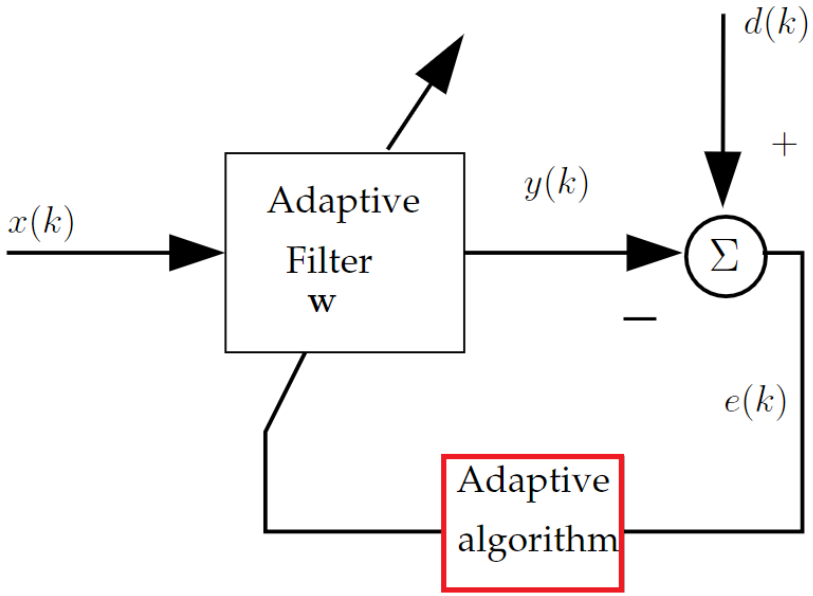
$n \rightsquigarrow$ word length.

$2^{-n+1} \rightsquigarrow$ Scale factor.

MAC $\rightsquigarrow$ Multiply accomulate.

$\circledast \rightsquigarrow$ Convolution.

Adaptive filter

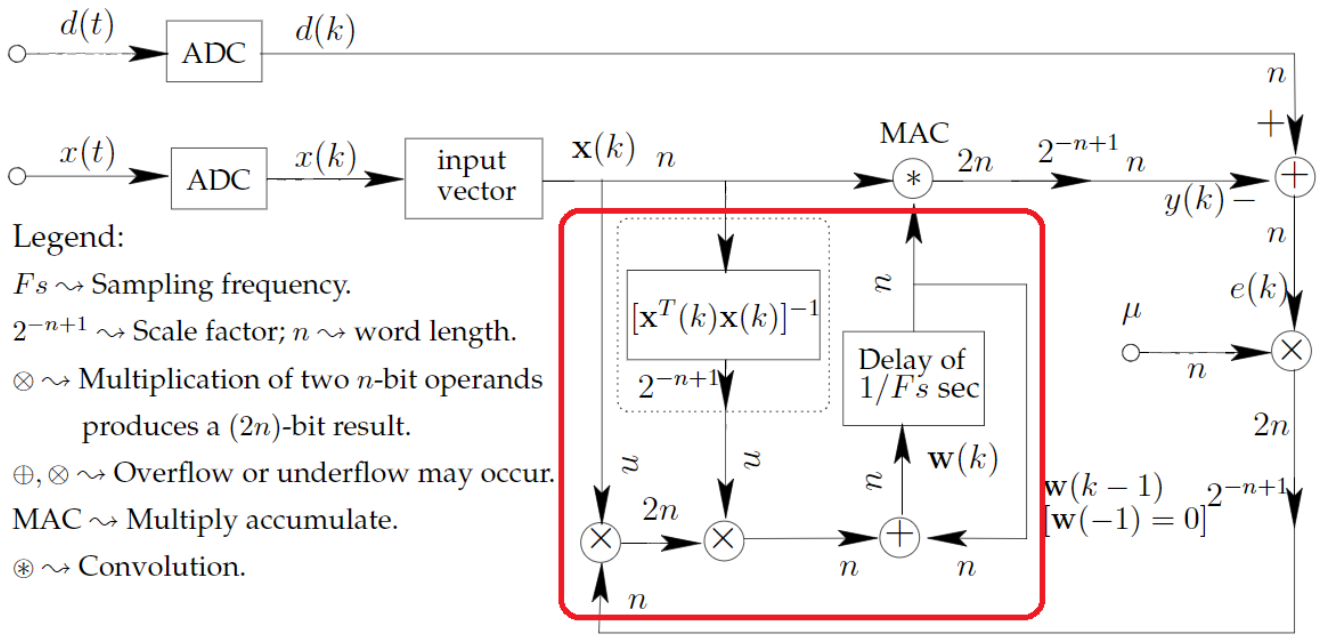


The updating equation of the NLMS algorithm expressed as

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \frac{\mu}{\epsilon(k) + \mathbf{x}^T(k)\mathbf{x}(k)}\mathbf{x}(k)e(k). \tag{5}$$

$\epsilon(k)$ is a small positive constant, keeping $\epsilon(k) = \epsilon$ leads to a so-called ϵ-NLMS algorithm ([1]).

Implement of NLMS adaptive filter



Median filter

$$M = \begin{cases} x(\frac{N-1}{2}) & \text{if } N \text{ is odd,} \\ \frac{1}{2}[x(\frac{N}{2}) + x(\frac{N}{2}+1)] & \text{if } N \text{ is even.} \end{cases} \tag{6}$$

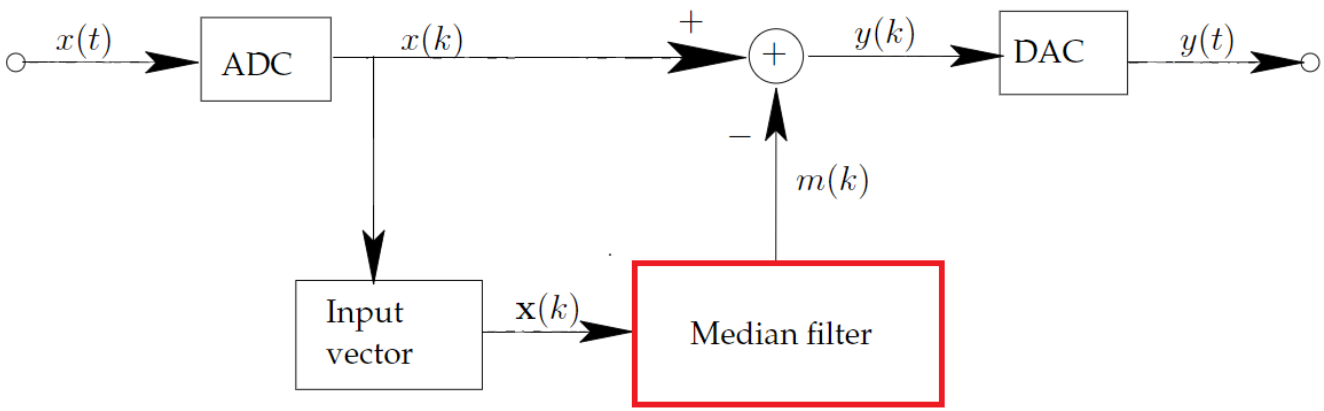
- $X = [1 \quad 2 \quad 2 \quad 6 \quad 7 \quad 9 \quad 10]$
  $N = 7$
  $M = 6$

Median filter

$$M = \begin{cases} x(\frac{N-1}{2}) & \text{if } N \text{ is odd,} \\ \frac{1}{2}[x(\frac{N}{2}) + x(\frac{N}{2}+1)] & \text{if } N \text{ is even.} \end{cases} \tag{6}$$

- $X = [1 \quad 2 \quad 2 \quad 6 \quad 7 \quad 9 \quad 10]$
  $N = 7$
  $M = 6$
- $X = [1 \quad 3 \quad 7 \quad 8]$
  $N = 4$
  $M = (3 + 7)/2 = 5$

Implement of Median filter



Experiments

NLMS adaptive filter-system identification

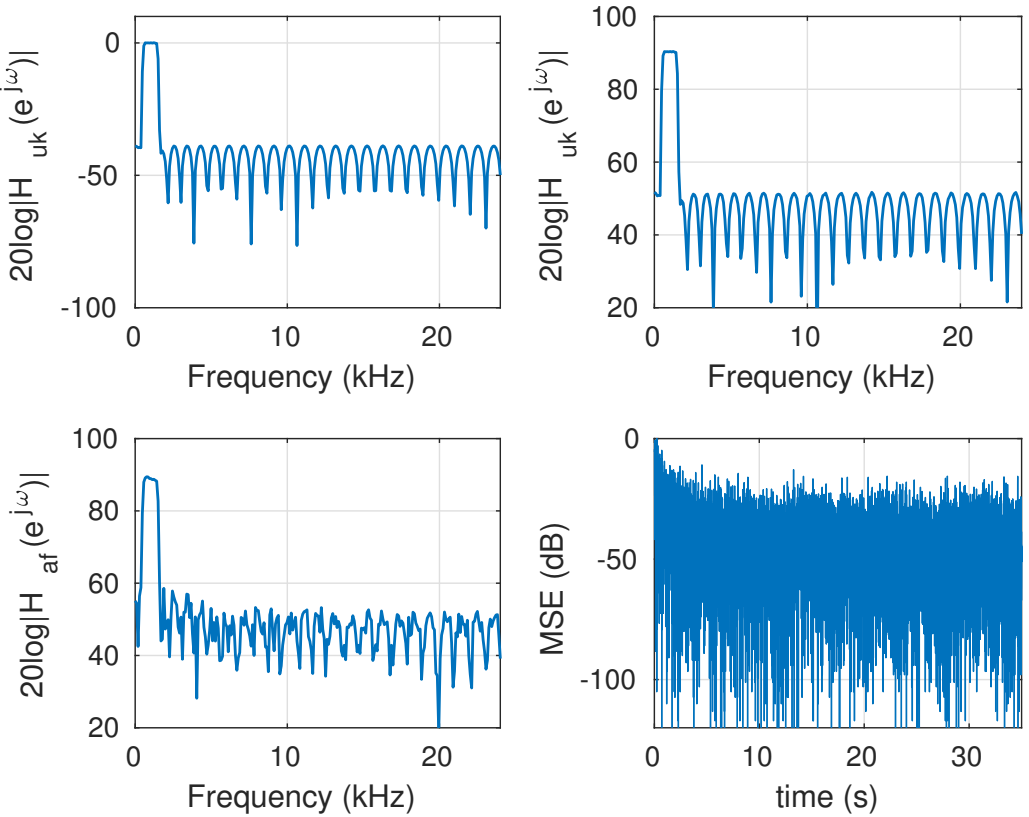


Figure 1: Fixed-point 16-bit implementation results using the NLMS adaptive filter to identify an 'unknown' passband filter with a stop band attenuation of 40 dB.

NLMS adaptive filter-propeller noise reduction

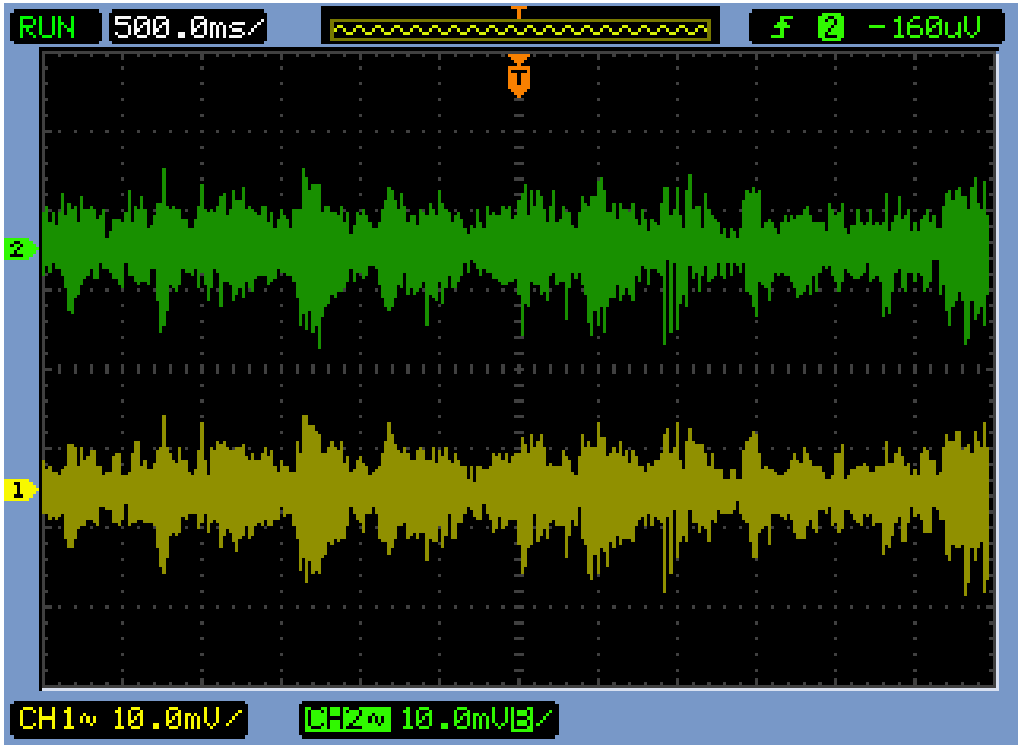


Figure 2: NLMS adaptive filter real-time comparison results.

Demonstration ▸ Link

FIR band-pass filter design and simulation

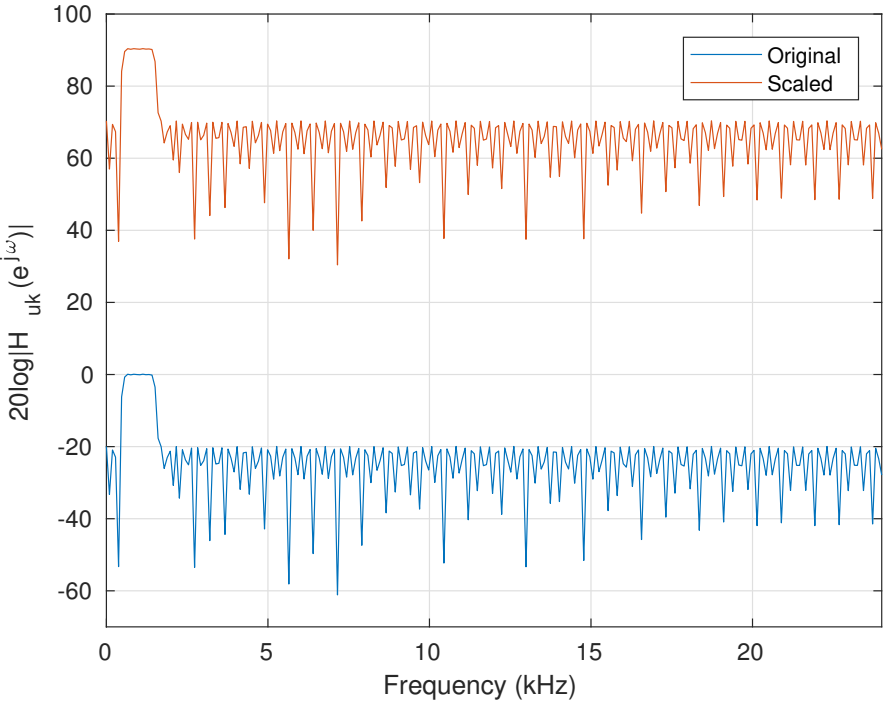


Figure 3: FIR frequency response.

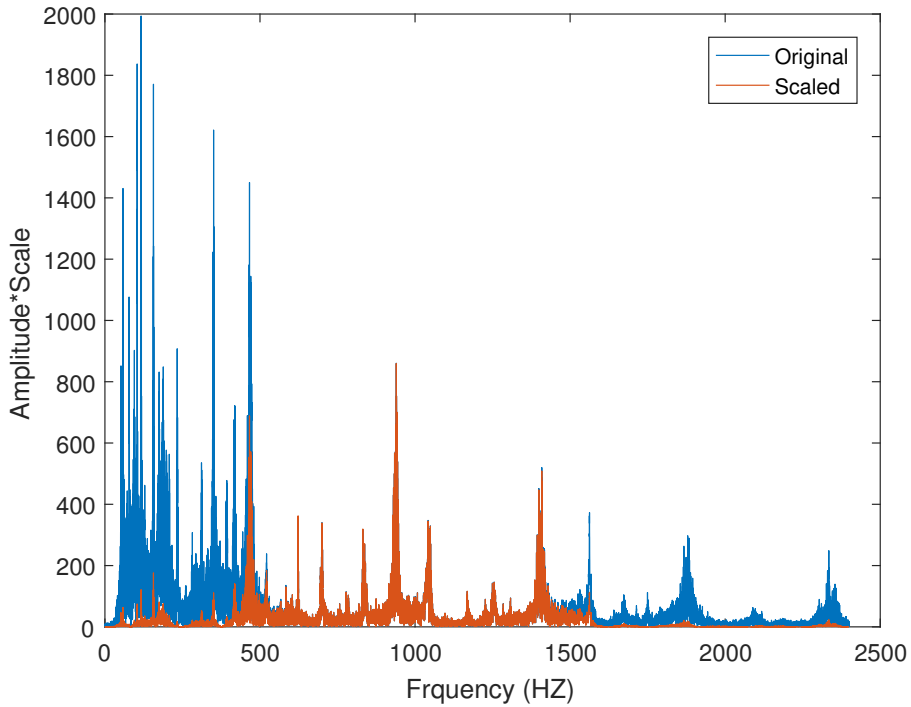


Figure 4: Audio samples comparisons on frequency domains.

FIR band-pass filter real-time comparison

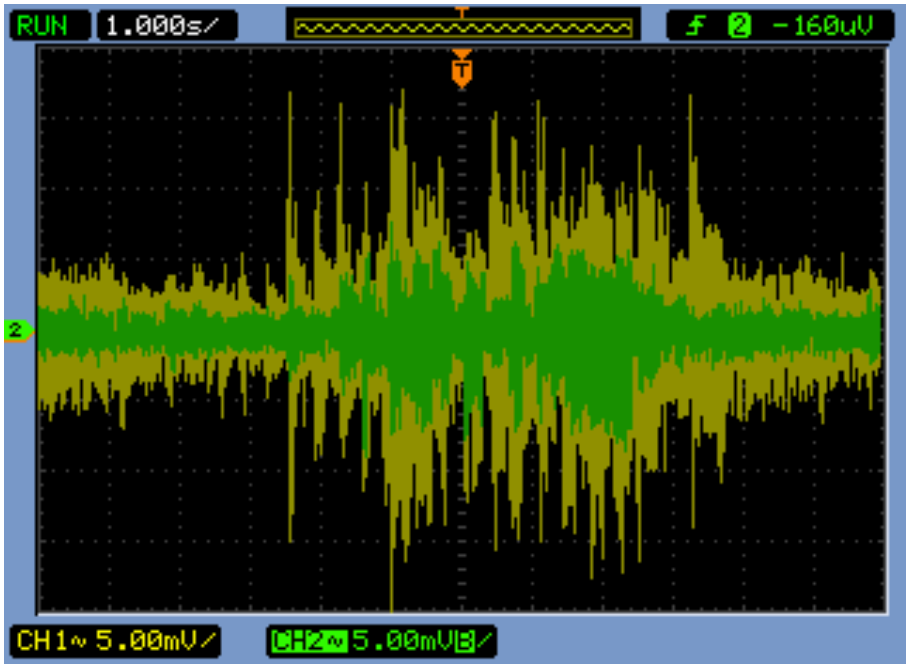


Figure 5: FIR original channel compared with real-time filtered channel.

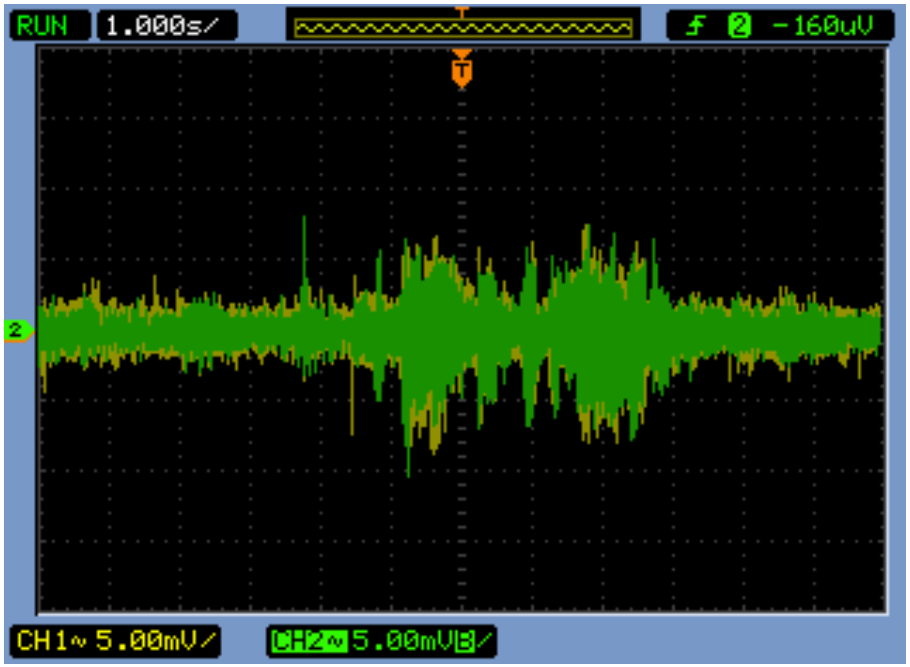


Figure 6: Comparison between filtered signal and real-time filtered signal.

Demonstration ▸ Link

Median Filter

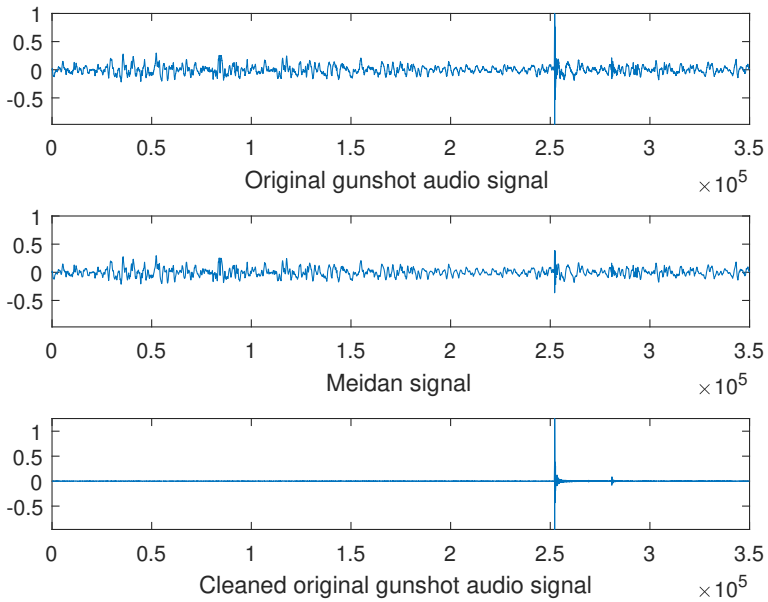


Figure 7: Gunshot audio signal cleaned by median filter.

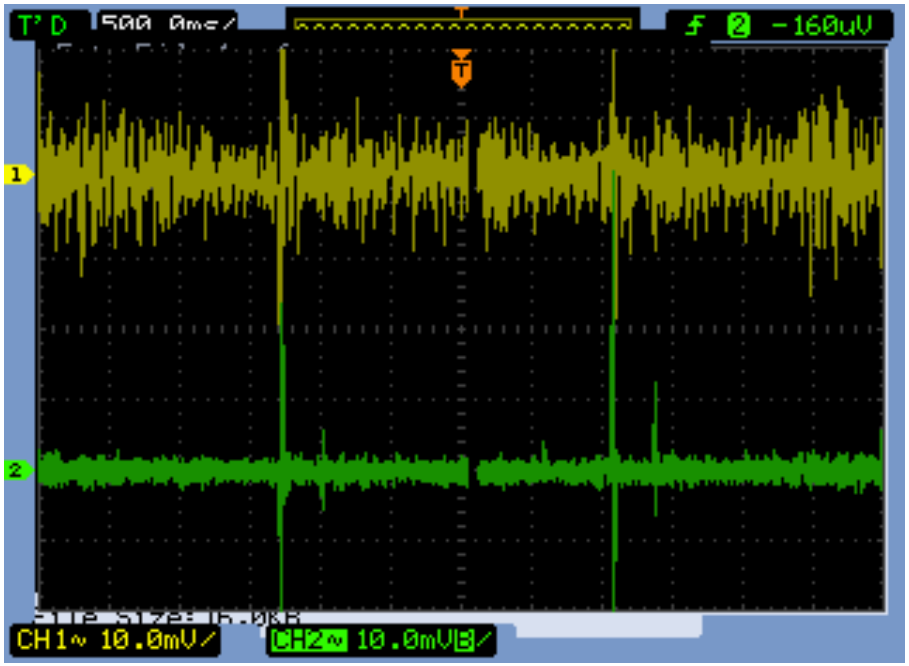


Figure 8: Median filter real-time comparison results.

Demonstration ▸ Link

Conclusion and Future work

Conclusion

- Significantly reduce propeller noise.

Conclusion

- Significantly reduce propeller noise.
- Be capable of frequency extraction.

Conclusion

- Significantly reduce propeller noise.
- Be capable of frequency extraction.
- Good performance on gunshot enhancement.

Conclusion

- Significantly reduce propeller noise.
- Be capable of frequency extraction.
- Good performance on gunshot enhancement.
- This platform is qualified for airborne real-time audio signal processing applications on UAVs.

Future Work

- More DSP applications on UAV.

Future Work

- More DSP applications on UAV.
- Integrate this design into autopilot.

Future Work

- More DSP applications on UAV.
- Integrate this design into autopilot.
- Implement DSP on FPGA for higher performance.

Future Work

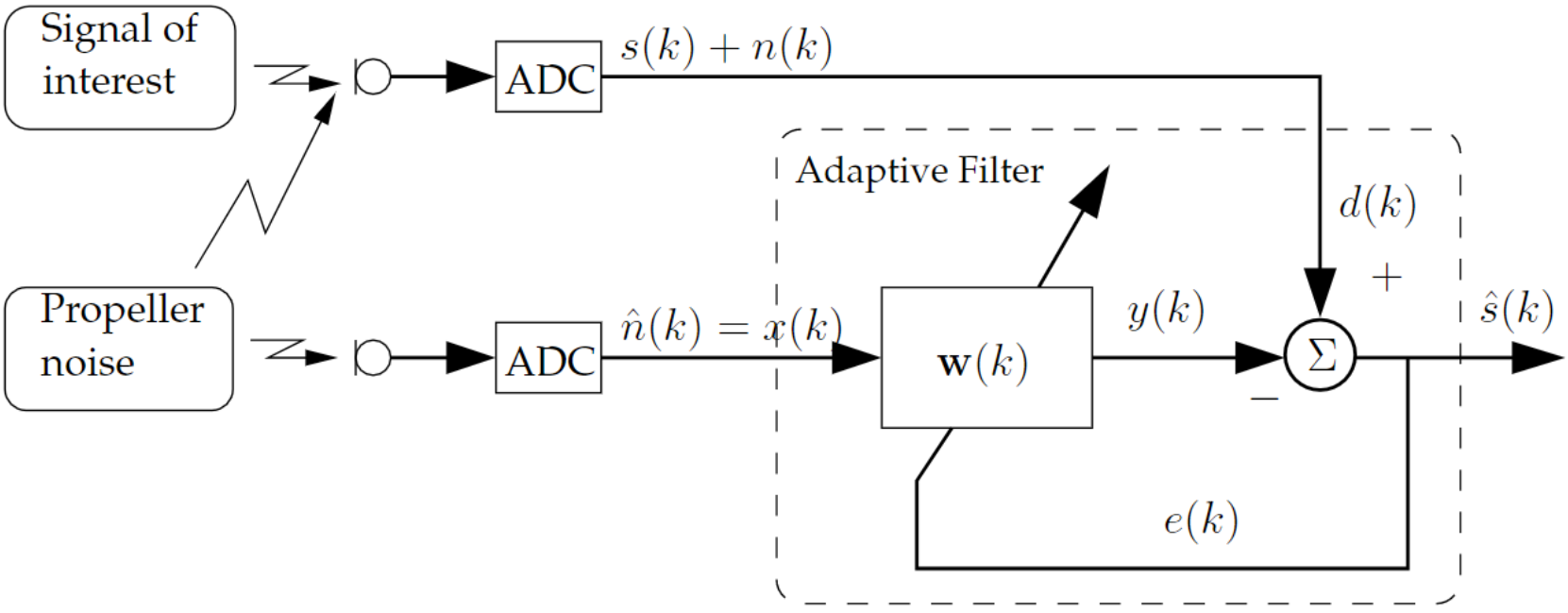
- More DSP applications on UAV.
- Integrate this design into autopilot.
- Implement DSP on FPGA for higher performance.
- Cooperate with UAV data transmission.

Questions?

References I


N. R. Yousef and A. H. Sayed.
A unified approach to the steady-state and tracking analyses of adaptive filters.
IEEE Transactions on Signal Processing, 49(2):314–324, Feb 2001.

Adaptive noise reduction setup



circular buffer

Listing 1: Software circular buffer.

```
typedef struct circular_buffer
{
    void *buffer_start; // data buffer
    void *buffer_end;   // end of data buffer
    size_t capacity;    // maximum number of items in the buffer
    size_t sz;          // size of each item in the buffer
    void *head;         // pointer to head
} cb;
```

Listing 2: Fixed-point FIR filter implemented by C Language.

```
//point inputp to the oldest data
if (cb_input->head == cb_input->buffer_end) {
  inputp = cb_input->buffer;
} else {
  inputp = cb_input->head;}
// perform the multiply-accumulate
for (k = 0; k < filterLength; k++) {
//when the input array pointer point to the end of buffer,
  jump to buffer start
  if (inputp == (cb_input->buffer_end)) {
    inputp = cb_input->buffer;
  }
  acc += (int32_t)(*coeffp++) * (int32_t)(*inputp++);
}
// saturate the result
if (acc > 0x3fffffff) {
  acc = 0x3fffffff;
} else if (acc < -0x40000000) {
  acc = -0x40000000;}
// convert from Q30 to Q15
return (int16_t)(acc >> 15);
```

Listing 3: Pseudocode for the 16-bit implementation of the NLMS algorithm.

```
for each new sample(k) do { acc = xx(k) = 0;
for ( i = 0; i < filterLength; i++){
  // MAC with scaling
    xx(k) +=  ((int32_t)(x[i]) * (int32_t)(x[i]))>>15;
    // MAC
  acc += (int32_t)((int32_t)*w[i]) * (int32_t)(x[i]);
}
  // Converting from Q30 to Q15
y(k) = (int16_t)(acc >> 15);
e(k) =   d(k) – y(k);
delta = (int32_t)((mu  *  (int32_t)e(k))>>15);
for(j = 0; j < filterLength; j++){
weights[j] += (int16_t)((delta * (int32_t)x[j])/(1 + xx(k)));}
 }
```

Listing 4: Median fetch function implemented by C Language.

```
int16_t median(int16_t n, int16_t* x) {
  int16_t temp;
  int16_t i, j;
    //sort the array x in ascending order
    for(i=0; i<n-1; i++) {
        for(j=i+1; j<n; j++) {
            if(x[j] < x[i]) {
                // swap elements
                temp = x[i];
                x[i] = x[j];
                x[j] = temp;
            }
        }
    }
    if(n%2==0) {
    // if there is an even number of elements, return mean of
    the two elements in the middle
        return((x[n/2] + x[n/2 - 1]) / 2);
    } else {
    // else return the element in the middle
        return x[n/2];}
}
```