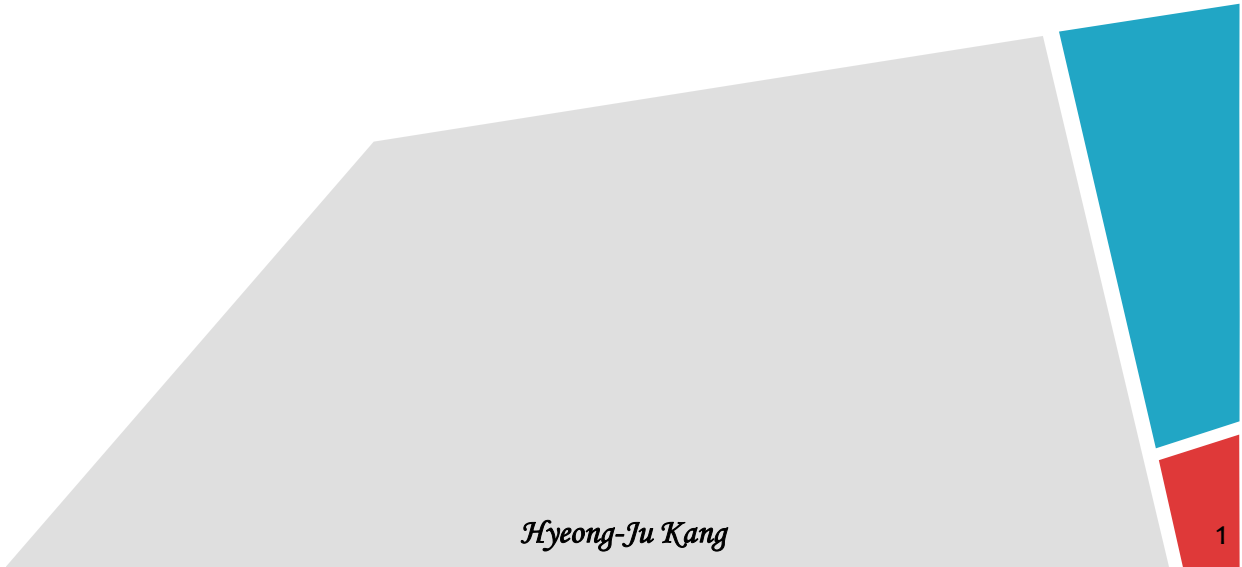




Lecture 1. 1D Filter



Outline



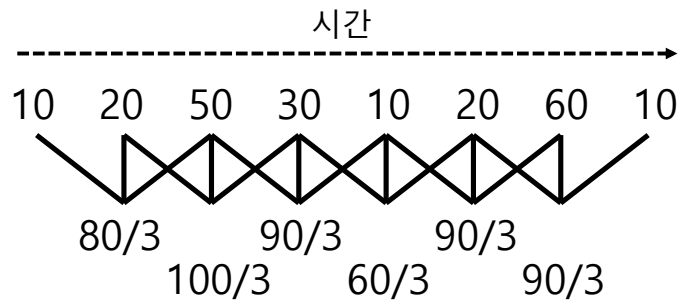
- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reconfigurable Design



Convolution?



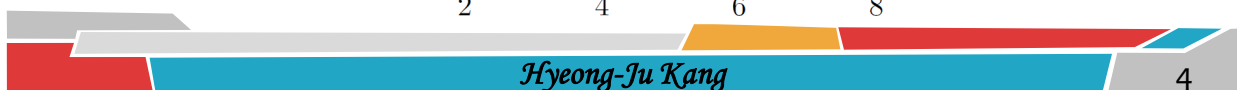
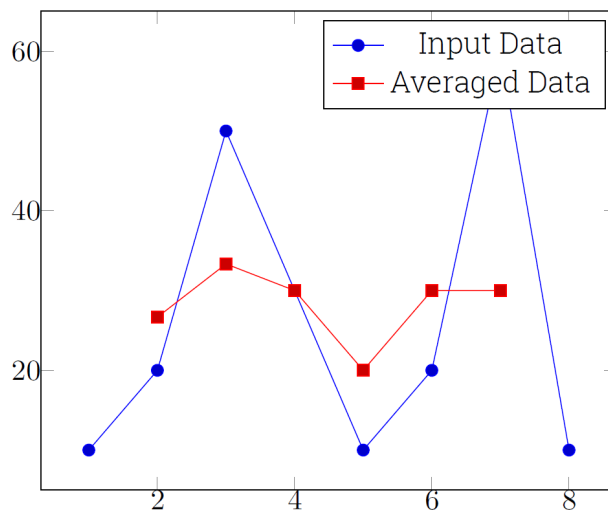
- ▶ 이동 평균(moving average)
 - ▶ 데이터를 따라 가며 평균 구하기



Convolution?

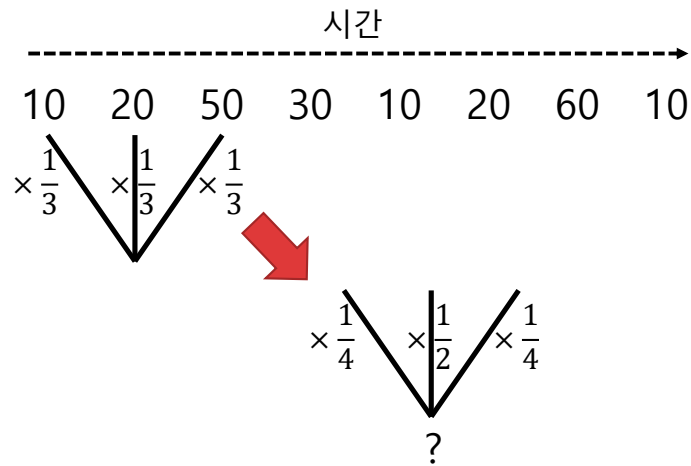


- ▶ 이동 평균(moving average)
 - ▶ 데이터를 따라 가며 평균 구하기
 - ▶ 그래프로 표현해 보면



▶ 이동 평균(moving average)

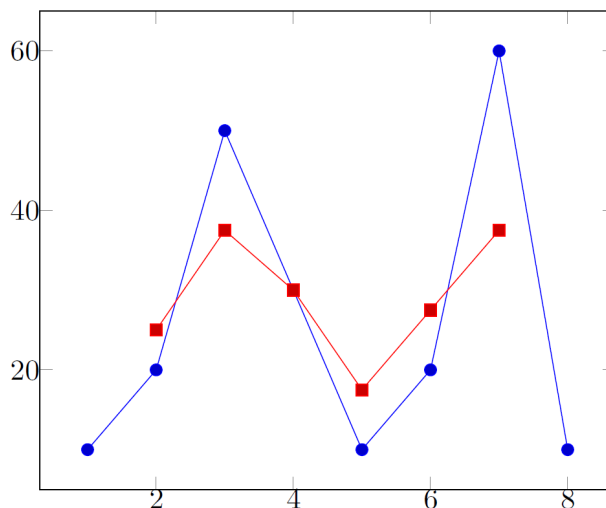
- ▶ 데이터를 따라 가며 평균 구하기



▶ 이동 평균(moving average)

- ▶ 데이터를 따라 가며 평균 구하기
- ▶ 그래프로 표현해 보면

$$\times \frac{1}{4} \quad \times \frac{1}{2} \quad \times \frac{1}{4}$$



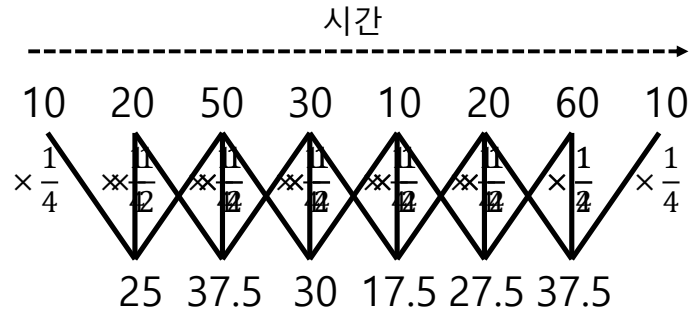


Convolution



▶ Convolution 연산

- ▶ 커널(kernel) $h = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$ 에 대한 Convolution 연산



- ▶ 계수(weight): 커널에 있는 값들



Convolution



▶ Convolution 연산

- ▶ 커널 $h = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$ 에 대한 Convolution 연산

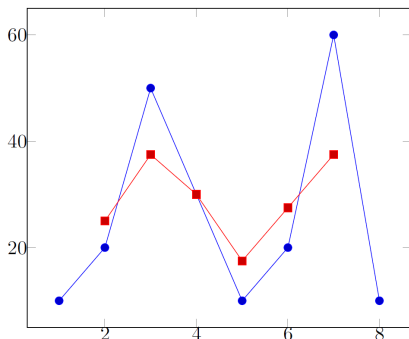
$$y[n] = h[-1] \times x[n-1] + h[0] \times x[n] + h[1] \times x[n+1]$$

- ▶ 일반적으로 커널 $h \left[-\frac{K-1}{2} \right] \sim h \left[\frac{K-1}{2} \right]$ 에 대해

$$y[n] = \sum_{k=-\frac{K-1}{2}}^{\frac{K-1}{2}} h[k] \times x[n+k]$$

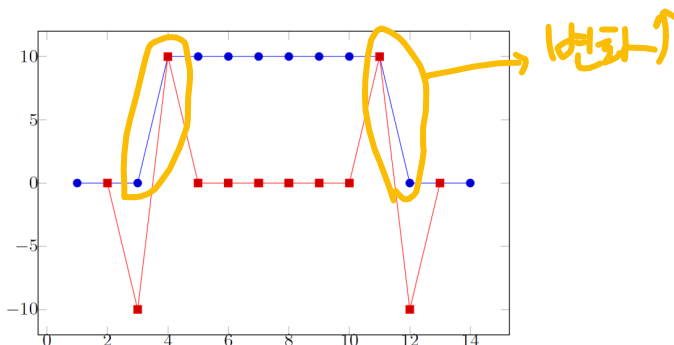


- ▶ 신호 처리에서 매우 많이 사용됨
- ▶ 주된 용도: Filtering
 - ▶ 원하는 주파수만 걸러냄
 - ▶ 앞의 예제는 low pass filter (LPF)의 예
 - ▶ 저역통과필터: 낮은 주파수 성분만 통과시키고 높은 주파수 성분은 제거



- ▶ 신호 처리에서 매우 많이 사용됨
- ▶ 주된 용도: Filtering
 - ▶ 원하는 주파수만 걸러냄
 - ▶ high pass filter (HPF)의 예
 - ▶ 고역통과필터: 높은 주파수 성분만 통과시키고 낮은 주파수 성분은 제거

$$h = [-1 \quad 2 \quad -1]$$





Outline



- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reconfigurable Design

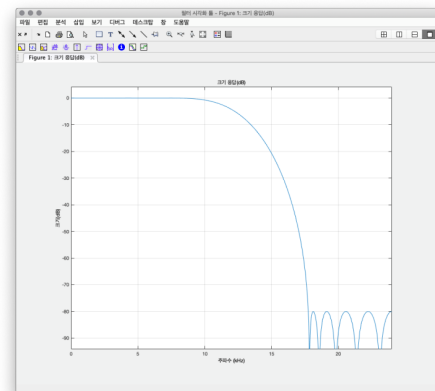


Filter의 설계



- ▶ 원하는 주파수 응답 모양에 맞게 Matlab 등을 이용하여 설계
 - ▶ ex) $N=21$, sampling rate 48kHz, passband 8kHz, passband ripple 0.01dB, stopband attenuation 80dB

```
N = 20;  
Fs = 48000;  
Fp = 8000;  
Ap = 0.01;  
Ast = 80;  
Rp = (10^(Ap/20) - 1)/(10^(Ap/20) + 1);  
Rst = 10^(-Ast/20);  
NUM = firceqrip(N,Fp/(Fs/2),[Rp Rst],'passedge');  
fvtool(NUM,'Fs',Fs)
```



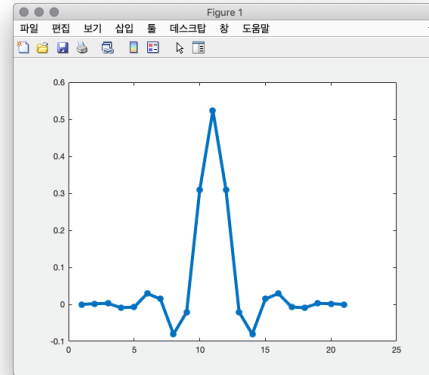


Filter의 설계



- ▶ 원하는 주파수 응답 모양에 맞게 Matlab 등을 이용하여 설계
 - ▶ ex) N=21, sampling rate 48kHz, passband 8kHz, passband ripple 0.01dB, stopband attenuation 80dB
- ▶ 설계의 결과는 Filter의 계수 $h[0] \sim h[N]$

```
-0.000306021779757585    0.00189666282674638
0.00257299891941012    -0.0090277441097318
-0.0075198095107301    0.0291019349476909
0.0145629522024416    -0.0807110457423653
-0.0210214194708373    0.308909103630076
0.523660422383779    0.308909103630076
-0.0210214194708373    -0.0807110457423653
0.0145629522024416    0.0291019349476909
-0.0075198095107301    -0.0090277441097318
0.00257299891941012    0.00189666282674638
-0.000306021779757585
```



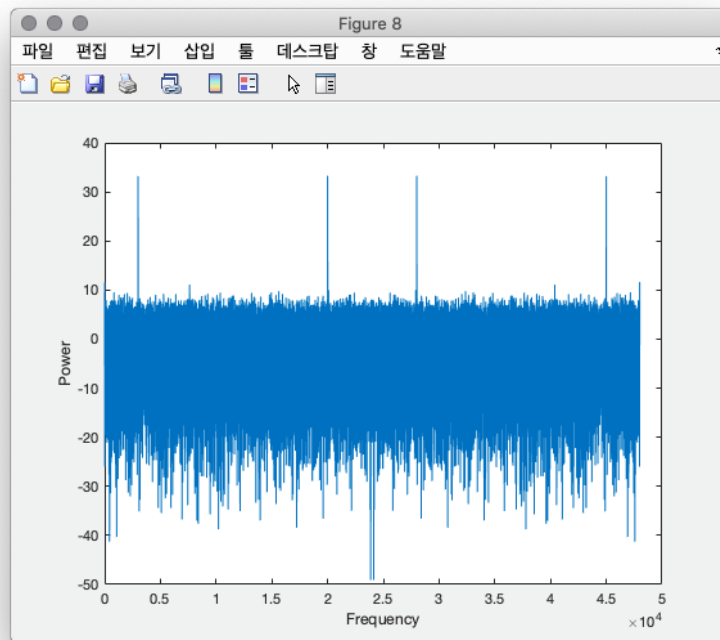
Filter 동작 확인



- ▶ 입력 데이터 생성
 - ▶ sampling rate 48kHz, 2 second span
 - ▶ 5kHz and 20kHz sine wave + random noise

```
t = 0:1/Fs:2-1/Fs; % 2 second span time vector
x = (0.3)*sin(2*pi*3000*t) ... % 5kHz sine
    + (0.3)*sin(2*pi*20000*t) ... % 20kHz sine
    + randn(size(t)); % random noise
plot(x);
xf = fft(x);
n = length(x); % number of samples
f = (0:n-1)*(Fs/n); % frequency range
xp = abs(xf).^2/n; % power of the DFT
plot(f,10*log10(xp))
xlabel('Frequency')
ylabel('Power')
save('filter_in.txt', 'x', '-ASCII');
```

▶ 입력 데이터 주파수 영역 파형

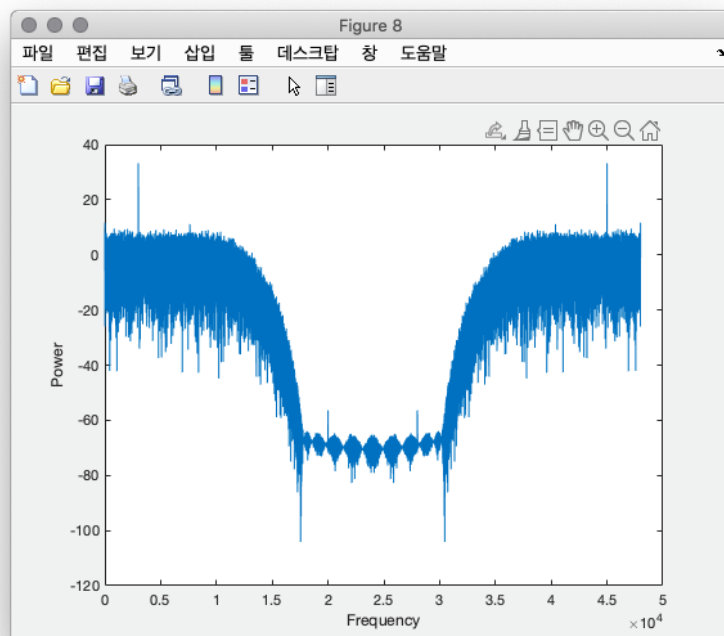


▶ Filter 결과 확인

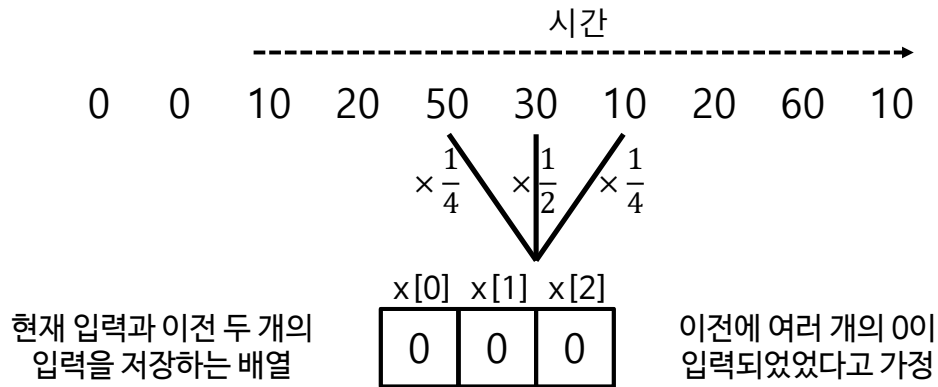
```

y = filter(NUM,1,x);
plot(y);
yf = fft(y);
yp = abs(yf).^2/n;
plot(f,10*log10(yp))
xlabel('Frequency')
ylabel('Power')

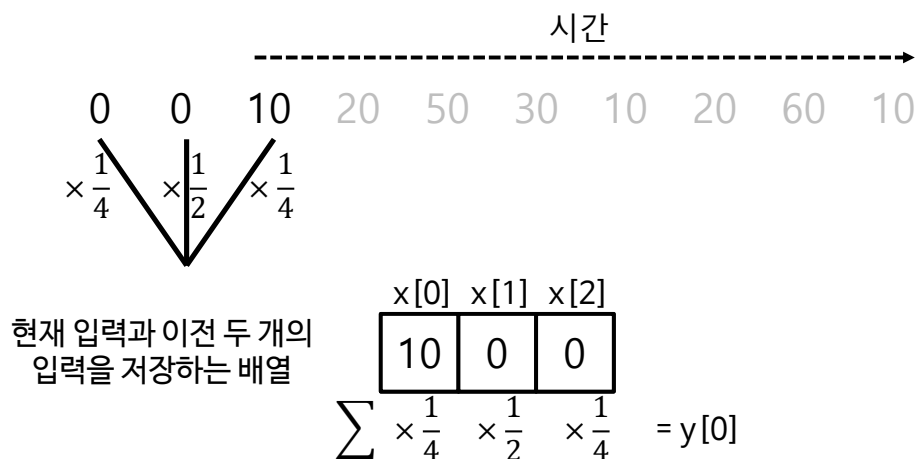
```



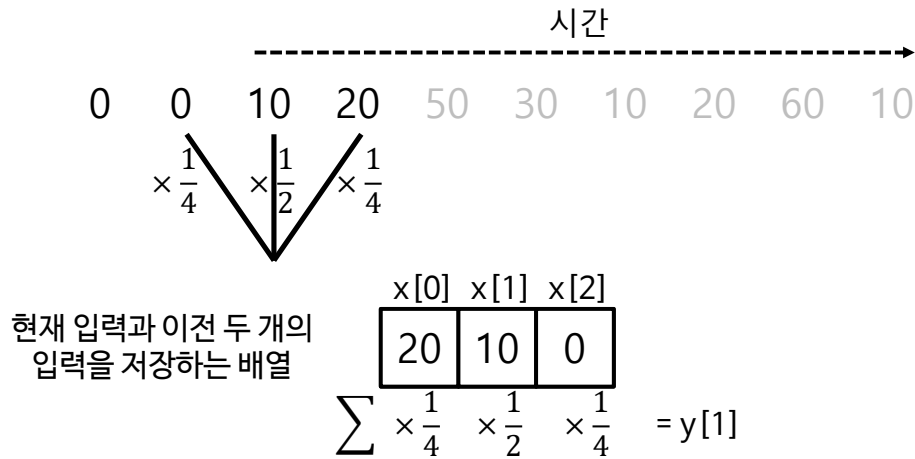
- ▶ 다양한 형태로 구현할 수 있으나 Filter기능을 담당하는 부분과 검증을 위한 부분을 구분하여 구현
- ▶ Filter 부분 구현 기본 개념



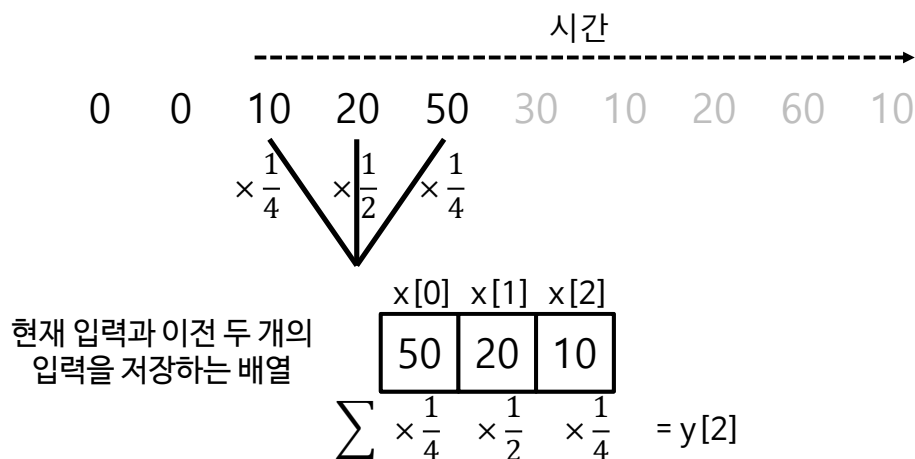
- ▶ 다양한 형태로 구현할 수 있으나 Filter기능을 담당하는 부분과 검증을 위한 부분을 구분하여 구현
- ▶ Filter 부분 구현 기본 개념

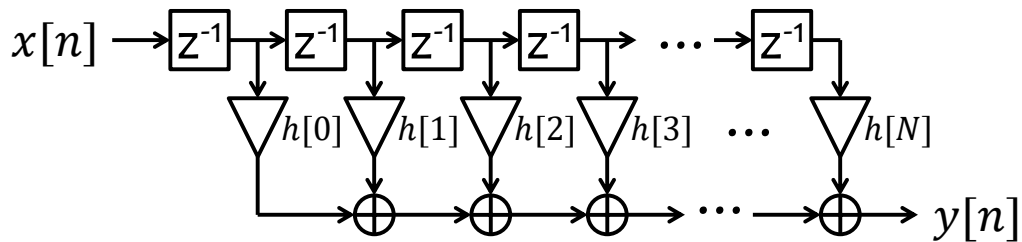


- ▶ 다양한 형태로 구현할 수 있으나 Filter기능을 담당하는 부분과 검증을 위한 부분을 구분하여 구현
- ▶ Filter 부분 구현 기본 개념



- ▶ 다양한 형태로 구현할 수 있으나 Filter기능을 담당하는 부분과 검증을 위한 부분을 구분하여 구현
- ▶ Filter 부분 구현 기본 개념





▶ N+1: 필터의 길이, tap의 개수

- ▶ 21-tap filter

▶ Filter 자체를 구현한 코드

```
float filter(float in) {
    static float x[21];

    double h[21] = { -0.000306021779757585, 0.00189666282674638,
                     0.00257299891941012, -0.0090277441097318,
                     ...,
                     -0.000306021779757585 };

    for(int i=20;i>=1;i--) {
        x[i] = x[i-1];
    }
    x[0] = in;

    float out = 0;
    for(int i=0;i<21;i++) {
        out += x[i] * h[i];
    }
    return out;
}
```

-----> 하나의 값을 입력 받아서
 -----> 현재 입력과 이전 20개의 입력값들을 저장하는 배열
 } 계수들
 } 기존에 저장된 값들을 한 칸씩 밀고
 새로운 입력 값을 x[0]에 저장
 } 저장된 입력값들과 계수들을 곱한 뒤 더하기
 -----> 합을 출력

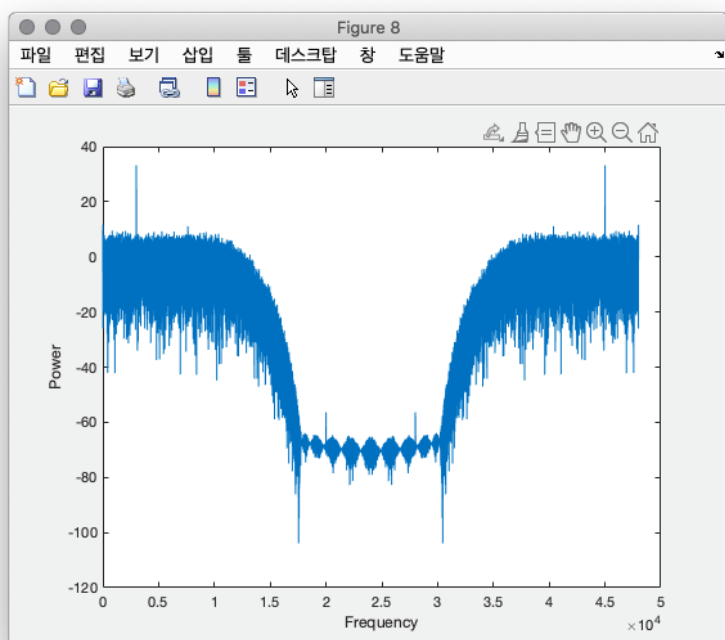
▶ Filter의 입출력을 담당하는 코드

```
int main(void) {
    FILE    *inf, *outf;
    inf = fopen("filter_in.txt", "r");
    outf = fopen("filter_out.txt", "w");

    float    in;
    while(fscanf(inf, "%f", &in) > 0) {
        float out;
        out = filter(in);
        fprintf(outf, "%f\n", out);
    }
    fclose(inf);
    fclose(outf);
}
```

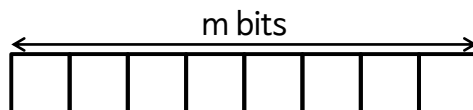
▶ 출력값을 저장한 파일을 매트랩에서 읽어서 주파수 영역 파형 확인

```
y = load('filter_out.txt');
n = length(y);
Fs = 48000;
f = (0:n-1)*(Fs/n);
yf = fft(y);
yp = abs(yf).^2/n;
plot(f,10*log10(yp))
xlabel('Frequency')
ylabel('Power')
```



- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reconfigurable Design

- ▶ Integer

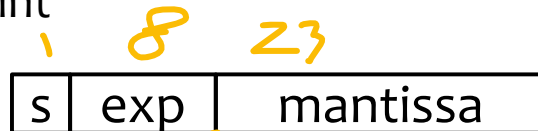


✓ 2'49

$$X = -x_{m-1}2^{m-1} + x_{m-2}2^{m-2} + \dots + x_12^1 + x_02^0$$

$$-2^{m-1} \leq X \leq 2^{m-1} - 1$$

- ▶ Floating-point



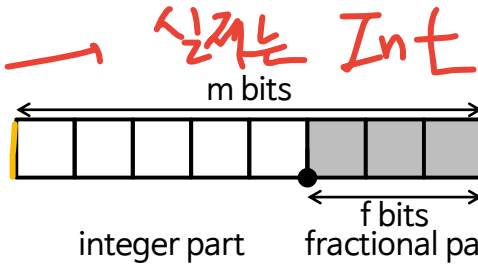
$$(-1)^s \times 2^{exp-bias} \times (1.mantissa)$$



Number Representation



Fixed-point



2's

$$X = -x_{m-1}2^{m-f-1} + \dots + x_f2^0 + x_{f-1}2^{-1} + \dots + x_02^{-f}$$

$$= \frac{-x_{m-1}2^{m-1} + x_{m-2}2^{m-2} + \dots + x_12^1 + x_02^0}{2^f}$$

$$\frac{-2^{m-1}}{2^f} \leq X \leq \frac{2^{m-1} - 1}{2^f}$$

코덱

>> f

*0 범위 제한
(2) 가상의 값*

- ▶ 형태는 integer와 같으나 scale (2^f)를 가지고 있음
 - ▶ 즉 2^f 배 된 상태로 사용하고 논리적으로만 scale을 고려함
 - ▶ ex) $m=8, f=3$ 에서, 0110 0111 은 103이지만 $103/8=12.875$ 을 의미함



Number Representation Conversion



Floating-point to Integer

```
float xf;
int xi;
...
xi = (int)floor(xf);
xi = (int)ceil(xf);
xi = (int)floor(xf+0.5);
```

m=8, f=2.01
f=3
Fl(2³ × 2.01)

Floating-point to Fixed-point

- ▶ integer로 변환하는 것과 동일하나 소수점 아래 f bit까지 표현 → 2^f 을 곱한 뒤 변환 (scale 2^f 는 논리적으로만)

```
float xf;
int xi;
...
xi = (int)floor(xf*(1<<f));
xi = (int)ceil(xf*(1<<f));
xi = (int)floor(xf*(1<<f)+0.5);
```

2^f 곱함

○○○Number Representation Conversion○○○

▶ Floating-point to Fixed-point

- ▶ integer로 변환하는 것과 동일하나 소수점 아래 f bit까지 표현 $\rightarrow 2^f$ 을 곱한 뒤 변환 (scale 2^f 는 논리적으로만)
- ▶ 표현 범위 $-2^{m-1} \leq X \leq 2^{m-1} - 1$

```
xi = (int)floor(xf*(1<<f)+0.5);  
  
if(xi >= 2m-1-1) xi = 2m-1-1;  
else if(xi <= -2m-1) xi = -2m-1;
```

- ▶ Note: 다음 범위는 논리적인 범위

$$\frac{-2^{m-1}}{2^f} \leq X \leq \frac{2^{m-1} - 1}{2^f}$$

○○○Number Representation Conversion○○○

▶ Floating-point to Fixed-point

- ▶ integer로 변환하는 것과 동일하나 소수점 아래 f bit까지 표현 $\rightarrow 2^f$ 을 곱한 뒤 변환 (scale 2^f 는 논리적으로만)
- ▶ 표현 범위 $-2^{m-1} \leq X \leq 2^{m-1} - 1 \rightarrow$ saturation

```
xi = (int)floor(xf*(1<<f)+0.5);  
  
if(xi >= 2m-1-1) xi = 2m-1-1;  
else if(xi <= -2m-1) xi = -2m-1;
```

- ▶ 양수/음수 대칭적인 범위를 원하면 (symmetric saturation)

$$-2^{m-1} + 1 \leq X \leq 2^{m-1} - 1$$

```
xi = (int)floor(xf*(1<<f)+0.5);  
  
if(xi >= 2m-1-1) xi = 2m-1-1;  
else if(xi <= -2m-1+1) xi = -2m-1+1;
```

▶ Fixed-Point Coefficients

▶ m = 16bit, f = 15bit 사용

▶ 계수들이 (-1, 1)의 범위에 있으므로 f = m-1로 정함

(-1, 1)

$$\frac{-2^{m-1}}{2^{m-1}} \leq X \leq \frac{2^{m-1} - 1}{2^{m-1}} \Rightarrow -1 \leq X \leq 1 - \frac{1}{2^{m-1}}$$

f=15

```
hi[i] = (int)floor(h[i]*32768+0.5);
```

2¹⁵(f)

▶ Fixed-Point Coefficients

▶ m = 16bit, f = 15bit 사용

```
hi[i] = (int)floor(h[i]*32768+0.5);
```

-10	62	84	-296	-246
954	477	-2645	-689	10122
17159	10122	-689	-2645	477
954	-246	-296	84	62
-10				

▶ 실제로는 1/2¹⁵을 곱한 값을 의미함

-0.0003	0.0019	0.0026	-0.0090	-0.0075
0.0291	0.0146	-0.0807	-0.0210	0.3089
0.5237	0.3089	-0.0210	-0.0807	0.0146
0.0291	-0.0075	-0.0090	0.0026	0.0019
-0.0003				



Fixed-Point Filter



▶ Fixed-Point Input

```
x = (0.3)*sin(2*pi*3000*t) ... % 5kHz sine
+ (0.3)*sin(2*pi*2000*t) ... % 20kHz sine
+ randn(size(t));           % random noise
```

▶ x의 범위는 대략 (-3.2, 3.2)

▶ randn 결과의 99%가 (-2.576, 2.576) 범위에 들어감

▶ m = 16bit, f = 13bit 사용

▶ integer part가 3bit이므로 [-4, 4) 표현 가능

$$\frac{-2^{m-1}}{2^{m-3}} \leq X \leq \frac{2^{m-1}-1}{2^{m-3}} \Rightarrow -4 \leq X \leq 4 - \frac{1}{2^{m-3}}$$

$f=13$

```
xi = (int)floor(x*8192+0.5);
```

2^{13}



c언어 구현



▶ Filter 자체를 구현한 코드

```
int filter(int in) {
    static int x[21];
    int h[21] = { -10, 62, 84, -296, -246,
                  954, 477, -2645, -689, 10122,
                  17159, 10122, -689, -2645, 477,
                  954, -246, -296, 84, 62,
                  -10};

    for(int i=20;i>=1;i--) {
        x[i] = x[i-1];
    }
    x[0] = in;

    int out = 0;
    for(int i=0;i<21;i++) {
        out += x[i] * h[i];
    }
    return out;
}
```

-----> 입력 자료형: 정수형

정수형 계수들

▶ Filter의 입출력을 담당하는 코드

```
int main(void) {
    FILE    *inf, *outf;
    inf = fopen("filter_in.txt", "r");
    outf = fopen("filter_out.txt", "w");

    float    in_f;
    while(fscanf(inf, "%f", &in_f) > 0) {
        int in_i = (int)floor(in_f*8192+0.5);
        if(in_i > 32767) in_i = 32767;
        else if(in_i < -32767) in_i = -32767;
        int out_i;
        float    out_f;
        out_i = filter(in_i);
        out_f = out_i / 8192.;
        fprintf(outf, "%f\n", out_f);
    }
    fclose(inf);
    fclose(outf);
}
```

-----> floating-point 데이터를 읽어서
 -----> fixed-point로 변환
 } symmetric saturation
 -----> fixed-point 입력
 -----> fixed-point 출력
 -----> floating-point로 변환
 -----> 원래 의미에 맞게 scale 적용
 -----> 8192???

▶ 두 m-bit integer를 곱하면?

$$-2^{m-1} \leq A \leq 2^{m-1} - 1 \quad -2^{m-1} \leq B \leq 2^{m-1} - 1$$

$$-2^{m-1} \cdot (2^{m-1} - 1) \leq A \cdot B \leq -2^{m-1} \cdot -2^{m-1}$$

$$-2^{2m-2} + 2^{m-1} \leq A \cdot B \leq 2^{2m-2}$$

▶ 2m bit으로 표현 가능

- ▶ 2m-1 bit으로는 표현 안 되는 경우 있음

$$-2^{2m-2} \leq X \leq 2^{2m-2} - 1$$

- ▶ 양수/음수 대칭이거나 -2^{m-1} 의 경우가 없음이 보장된다면 2m-1 bit으로 표현 가능



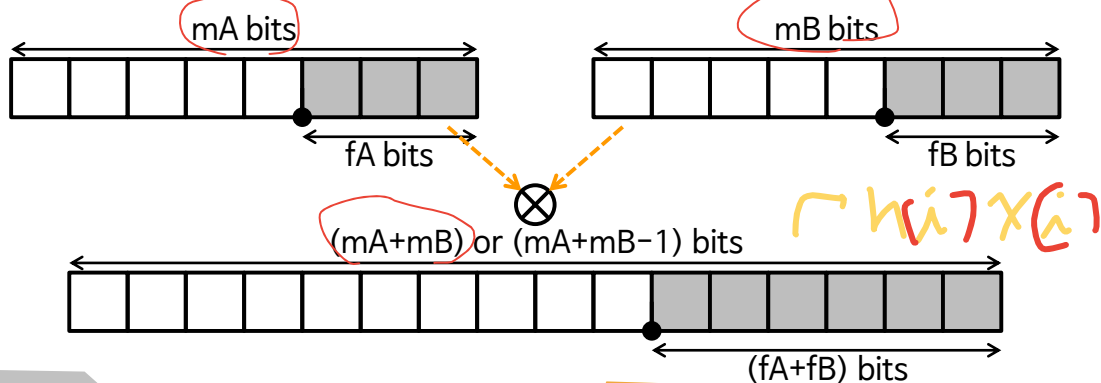
곱셈 결과의 범위



- ▶ 두 m, f fixed-point를 곱하면?

$$\frac{-2^{m_A-1}}{2^{f_A}} \leq A \leq \frac{2^{m_A-1}-1}{2^{f_A}} \quad \frac{-2^{m_B-1}}{2^{f_B}} \leq B \leq \frac{2^{m_B-1}-1}{2^{f_B}}$$

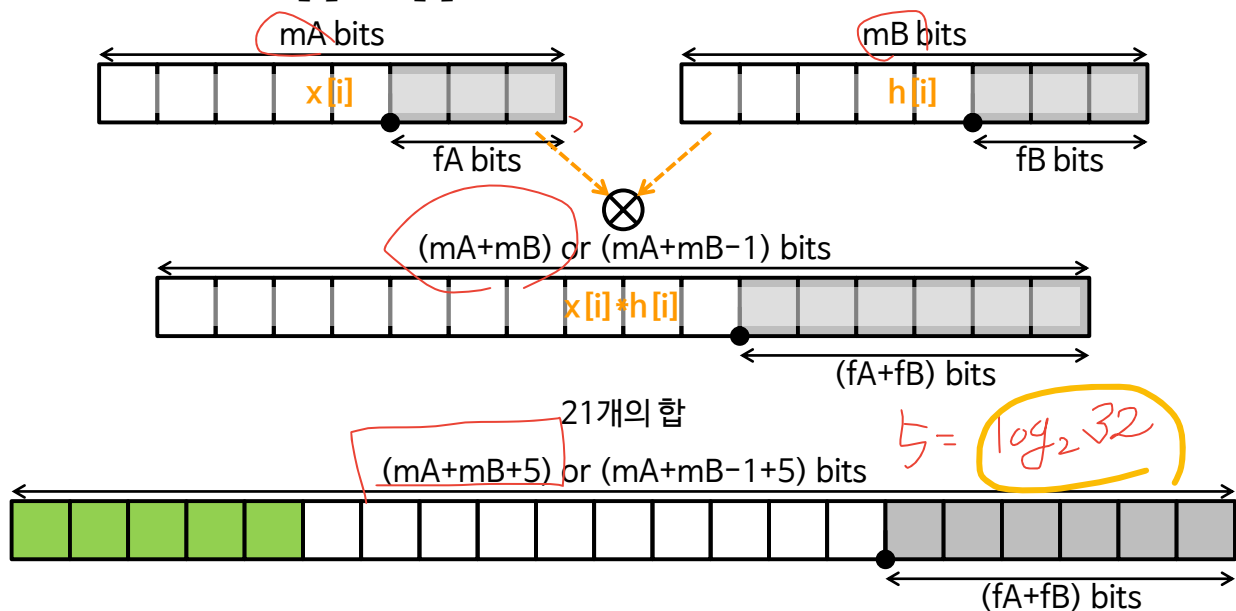
$$\frac{-2^{m_A+m_B-2} + 2^{m_A-1}}{2^{f_A+f_B}} \leq A \cdot B \leq \frac{2^{m_A+m_B-2}}{2^{f_A+f_B}}$$



Filter 계산 결과



- ▶ 21개의 $x[i] * h[i]$ 을 더함





Filter 계산 결과와 출력



▶ Filter 계산 결과 정리 (m,f)

▶ $x[i] * h[i] : (16,13) * (16,15) \rightarrow (31,28)$

▶ $x[i]$ 와 $h[i]$ 에 symmetric saturation 가정

▶ $\Sigma x[i]*h[i] : (36, 28)$

▶ 출력을 입력과 같이 16bit로 한다면?



▶ (16,8)로 표현

▶ 하위 비트들은? truncation or rounding

GRS?

```
out = out >> 20;
```

```
out = (out+(1<<19)) >> 20;
```



Filter 계산 결과와 출력



▶ Filter 계산 결과 정리 (m,f)

▶ $x[i] * h[i] : (16,13) * (16,15) \rightarrow (31,28)$

▶ $x[i]$ 와 $h[i]$ 에 symmetric saturation 가정

▶ $\Sigma x[i]*h[i] : (32,28)$

▶ Filter의 특성상 계수의 절대값의 합 만큼만 커짐 $\rightarrow 1.5$ 배

▶ 출력을 입력과 같이 16bit로 한다면?



▶ (16,12)로 표현

▶ 하위 비트들은? truncation or rounding

$$32 - 16 = 16$$

```
out = out >> 16;
```

```
out = (out+(1<<15)) >> 16;
```

▶ Filter 자체를 구현한 코드

```

int filter(int in) {
    static int x[21];
    int h[21] = { -10, 62, 84, -296, -246,
                  954, 477, -2645, -689, 10122,
                  17159, 10122, -689, -2645, 477,
                  954, -246, -296, 84, 62,
                  -10};
    for(int i=20;i>=1;i--) {
        x[i] = x[i-1];
    }
    x[0] = in;
    int out = 0;
    for(int i=0;i<21;i++) {
        out += x[i] * h[i];
    }
    out = (out+(1<<15)) >> 16;
    if(out > 32767) out = 32767;
    else if(out < -32767) out = -32767;
    return out;
}

```

▶ Filter의 입출력을 담당하는 코드

```

int main(void) {
    FILE *inf, *outf;
    inf = fopen("filter_in.txt", "r");
    outf = fopen("filter_out.txt", "w");

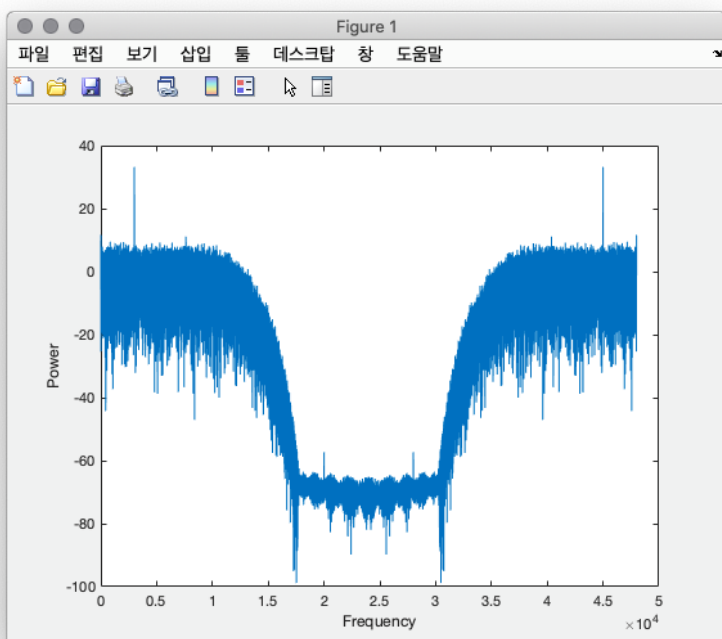
    float in_f;
    while(fscanf(inf, "%f", &in_f) > 0) {
        int in_i = (int)floor(in_f*8192+0.5);
        if(in_i > 32767) in_i = 32767;
        else if(in_i < -32767) in_i = -32767;
        int out_i;
        float out_f;
        out_i = filter(in_i);
        out_f = out_i / 4096;
        fprintf(outf, "%f\n", out_f);
    }
    fclose(inf);
    fclose(outf);
}

```

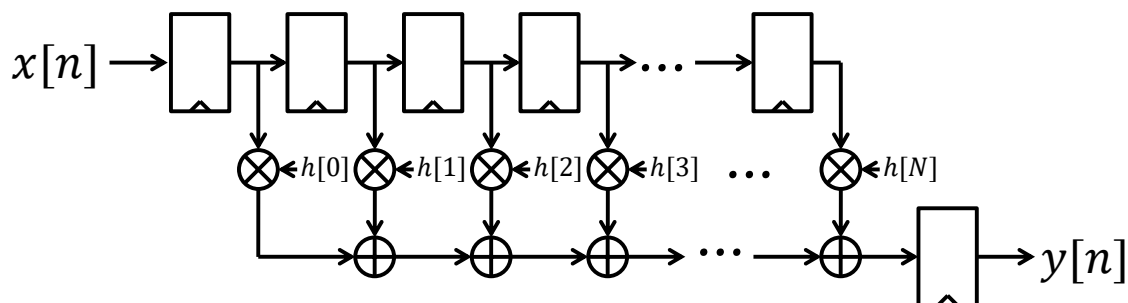
floating-point로 변환
원래 의미에 맞게 scale 적용
(16,12)이므로 1/4096

- ▶ 출력값을 저장한 파일을 매트랩에서 읽어서 주파수 영역 파형 확인

```
y = load('filter_out.txt');
n = length(y);
Fs = 48000;
f = (0:n-1)*(Fs/n);
yf = fft(y);
yp = abs(yf).^2/n;
plot(f,10*log10(yp))
xlabel('Frequency')
ylabel('Power')
```



- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reconfigurable Design



```

module filter (
    input    clk,
    input    n_reset,
    input    [15:0] x_in,
    output   reg [15:0] y_out
);

reg signed [15:0] x[20:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        x[0] <= 'b0;
        ...
        x[20] <= 'b0;
    end else begin
        x[0] <= x_in;
        x[1] <= x[0];
        ...
        x[20] <= x[19];
    end
end

end

```



Verilog 구현



```

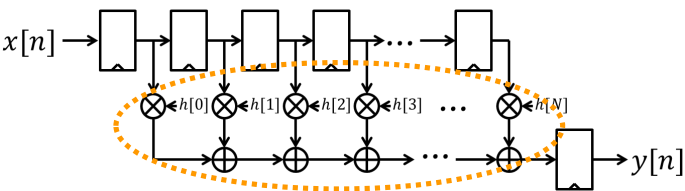
reg signed [15:0] h[20:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[0] <= -16'd 10;
        h[1] <= 16'd 62;
        ...
        h[20] <= -16'd 10;
    end else begin
    end
end

wire signed [31:0] mul[20:0]; // (16,13) * (16,15) -> (31,28) if sym. sat.
// (32, 28) sum

assign mul[0] = x[0] * h[0];
assign mul[1] = x[1] * h[1];
...
assign mul[20] = x[20] * h[20];

wire signed [31:0] sum = mul[0] + mul[1] + mul[2] + ...;

```




Verilog 구현



```

wire signed [31:0] mul[20:0]; // (16,13) * (16,15) -> (31,28) if sym. sat.
// (32, 28) sum

assign mul[0] = x[0] * h[0];
assign mul[1] = x[1] * h[1];
...
assign mul[20] = x[20] * h[20];

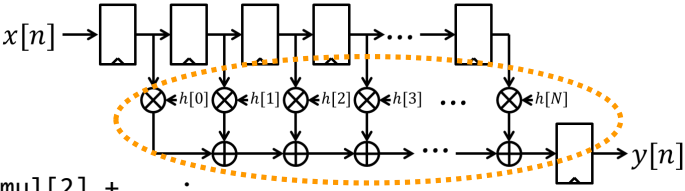
wire signed [31:0] sum;
assign sum = mul[0] + mul[1] + mul[2] + ...;

wire signed [31:0] y_rnd = sum + (1<<15); // for rounding

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        y_out <= 'b0;
    end else begin
        y_out <= y_rnd[31:16];
    end
end

endmodule

```



- ▶ 어떻게 검증할 것인가?
 - ▶ C 모델과 동일한 입력을 넣은 뒤
 - ▶ 출력이 서로 같은지 비교

```
fixed_in = fopen("filter_in_fixed.txt", "w");
fixed_out = fopen("filter_out_fixed.txt", "w");
```

```
float    in_f;
while(fscanf(in_f, "%f", &in_f) > 0) {
    int in_i = (int)floor(in_f*8192+0.5);
    if(in_i > 32767) in_i = 32767;
    else if(in_i < -32767) in_i = -32767;
    fprintf(fixed_in, "%04X\n", in_i & 0xFFFF);
    int out_i;
    float out_f;
    out_i = filter(in_i);
    fprintf(fixed_out, "%11d\n", out_i);
    out_f = out_i / 4096.;
    fprintf(out_f, "%f\n", out_f);
}
```

16-bit mask

입출력 값을 파일에 저장

```
module top_filter;

reg clk, n_reset;
initial clk = 1'b0;
always #5 clk = ~clk;

reg [15:0] in_data[0:95999];
reg [15:0] x_in;
wire signed [15:0] y_out;
integer idx;
initial begin
    n_reset = 1'b1;
    idx = 0;
    x_in = 0;
    $readmemh("../c/filter_in_fixed.txt", in_data);
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
end
```

입력 데이터를 저장할 곳

readmemh에 적합하게 파일을 구성해야 함

C 모델에서 입력 값들을 저장한 파일에서 데이터 읽기



Testbench



```

@(posedge clk);
@(posedge clk);
@(posedge clk);
for(idx=0;idx<96000;idx=idx+1) begin
    x_in = in_data[idx]; -----> 파일에서 읽은 값을 차례로 x_in으로 공급
    @(posedge clk);
end
x_in = 0;
@(posedge clk);
@(posedge clk);
@(posedge clk);
$finish;
end

filter i_filter (.clk(clk), .n_reset(n_reset), .x_in(x_in), .y_out(y_out));

always@(posedge clk) begin
    $display("%d", y_out); -----> 결과값을 화면에 출력
end

endmodule

```



결과 비교



- ▶ 결과 비교는 어떻게?
 - ▶ 눈으로?
 - ▶ 화면 출력을 파일로 받은 뒤 diff
 - ▶ ./sim.v > result
 - ▶ C에서 출력 값 파일 구성할 때 \$display 출력에 맞추어서


```
fprintf(fixed_out, "%11d\n", out_i);
```
 - ▶ 실시간 비교
 - ▶ 실시간으로 비교해야 오동작이 발생한 그 순간을 파악하기 쉬움 → 검증과 디버깅에서 중요함
 - ▶ 방법1: 출력 값도 testbench에서 readmemh로 읽은 뒤 하나씩 비교
 - ▶ 주의: latency, x value 출력
 - ▶ 방법2: Direct Programming Interface (DPI) → 다음 시간에



실시간 비교



```

module top_filter;

reg clk, n_reset;
initial clk = 1'b0;
always #5 clk = ~clk;

reg [15:0] in_data[0:95999];
reg [15:0] out_data[0:95999];

reg [15:0] x_in;
wire signed [15:0] y_out;
integer idx, out_idx;
initial begin
    n_reset = 1'b1;
    idx = 0; out_idx = 0;
    x_in = 0;
    $readmemh("../c/filter_in_fixed.txt", in_data);
    $readmemh("../c/filter_out_fixed.txt", out_data);
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
end

```

출력 데이터를 저장할 곳

몇 번째 출력 데이터와 비교?

readmemh에 적합하게 파일을 구성해야 함

C 모델에서 출력 값들을 저장한 파일에서 데이터 읽기



실시간 비교



```

filter i_filter (.clk(clk), .n_reset(n_reset), .x_in(x_in), .y_out(y_out));

always@(posedge clk) begin
    if(out_idx >= 6) begin
        if(out_data[out_idx-6] != y_out) begin
            $display("Error!!");
            #10; $finish;
        end
    end
    out_idx = out_idx + 1;
end
endmodule

```

latency 고려

C 모델 출력과 다른 결과가 나오면 Error를 출력하고 종료

주의: 한 쪽이 unknown(x)이면 조건은 false y_out이 x는 아닌지 확인 필요

그 다음 데이터와 비교



Using Loop



```
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        x[0] <= 'b0;
        ...
        x[20] <= 'b0;
    end else begin
        x[0] <= x_in;
        x[1] <= x[0];
        ...
        x[20] <= x[19];
    end
end
```



```
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<21;i++) begin
            x[i] <= 'b0;
        end
    end else begin
        x[0] <= x_in;
        for(int i=1;i<21;i++) begin
            x[i] <= x[i-1];
        end
    end
end
```



Using Loop



```
assign mul[0] = x[0] * h[0];
assign mul[1] = x[1] * h[1];
...
assign mul[20] = x[20] * h[20];

wire signed [31:0] sum;
assign sum = mul[0] + mul[1] + mul[2] + ...;
```

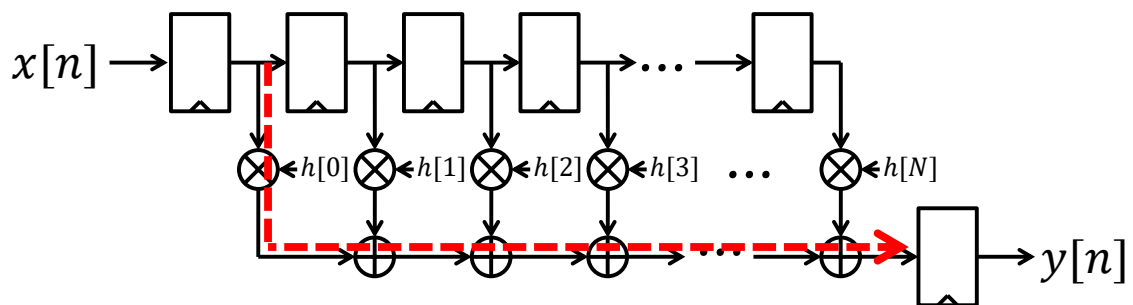


```
genvar i;
for(i=0;i<21;i++) begin
    assign mul[i] = x[i] * h[i];
end

reg signed [31:0] sum;
always@(*) begin
    sum = 0;
    for(int i=0;i<21;i++) begin
        sum = sum + mul[i];
    end
end
```

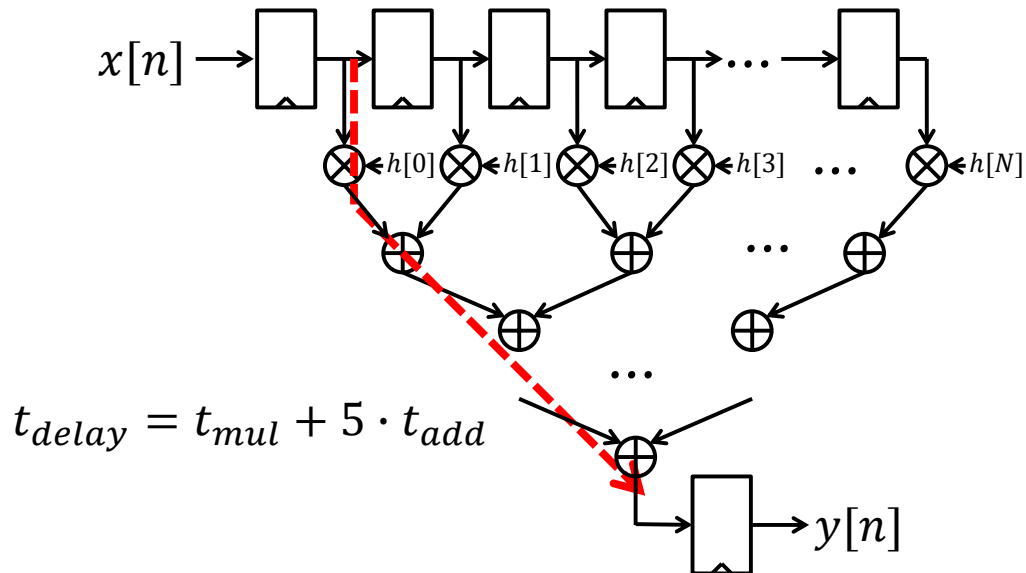
- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reconfigurable Design

- ▶ 예상 critical path – naïve implementation



$$t_{delay} = t_{mul} + 20 \cdot t_{add}$$

▶ 예상 critical path – adder tree



▶ 이 RTL 기술에서 adder tree를 합성할 수 있는가?

```

wire signed [31:0] sum;
assign sum = mul[0] + mul[1] + mul[2] + ...;

```

```

reg signed [31:0] sum;
always@(*) begin
    sum = 0;
    for(int i=0;i<21;i++) begin
        sum = sum + mul[i];
    end
end

```

▶ 만일 못 하면? → 명시적으로 기술 → 이것은 어떻게?

```

wire signed [31:0] sum;
assign sum = ((((((mul[0] + mul[1]) + (mul[2] + mul[3]))
                + ((mul[4] + mul[5]) + (mul[6] + mul[7]))))
                ...
                + mul[20]))))));

```

합성 결과 – Critical Path Delay

Point	Incr	Path
h_reg[19][9]/CK (SEL_FDPRBQ_D_1P5)	0.00	0.00 r
h_reg[19][9]/Q (SEL_FDPRBQ_D_1P5)	0.10	0.10 r
mult_69_G20/B[9] (filter_DW02_mult_19)	0.00	0.10 r
...		
mult_69_G20/PRODUCT[23] (filter_DW02_mult_19)	0.00	1.95 f
add_15_root_add_0_root_add_80/B[23] (filter_DW01_add_107)	0.00	1.95 f
...		
add_15_root_add_0_root_add_80/SUM[24] (filter_DW01_add_107)	0.00	2.21 r
add_9_root_add_0_root_add_80/B[24] (filter_DW01_add_64)	0.00	2.21 r
...		
add_9_root_add_0_root_add_80/SUM[26] (filter_DW01_add_64)	0.00	2.43 f
add_4_root_add_0_root_add_80/B[26] (filter_DW01_add_203)	0.00	2.43 f
...		
add_4_root_add_0_root_add_80/SUM[29] (filter_DW01_add_203)	0.00	2.70 f
add_1_root_add_0_root_add_80/A[29] (filter_DW01_add_50)	0.00	2.70 f
...		
add_1_root_add_0_root_add_80/SUM[31] (filter_DW01_add_50)	0.00	2.94 f
add_0_root_add_0_root_add_80/B[31] (filter_DW01_add_52)	0.00	2.94 f
...		
add_0_root_add_0_root_add_80/SUM[31] (filter_DW01_add_52)	0.00	3.03 f
y_out_reg[15]/D (SEL_FDPRBQ_D_1P5)	0.00	3.03 f
data arrival time		3.03

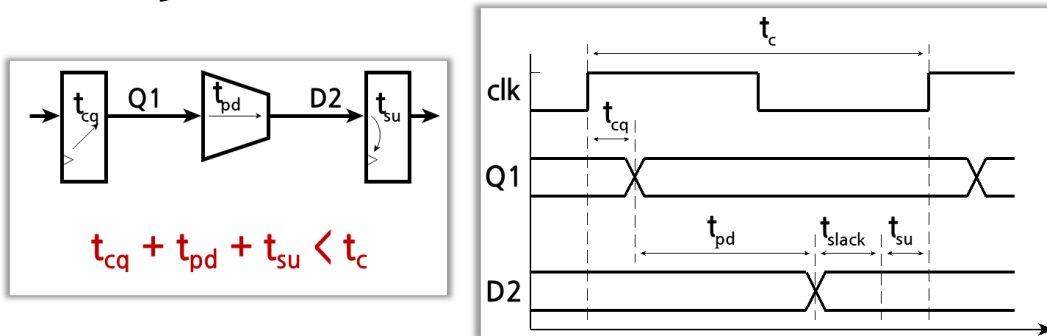
Hyeong-Ju Kang

61

Clock Frequency

▶ clock frequency

- ▶ clock-q 0.1ns + path delay 2.93ns + set-up time 0.04ns = 3.07ns
- ▶ → 326MHz



▶ 더 빨리 동작시키고 싶으면? → pipeline

Hyeong-Ju Kang

62

합성 결과 – Critical Path Delay

Point	Incr	Path
h_reg[19][9]/CK (SEL_FDPRBQ_D_1P5)	0.00	0.00 r
h_reg[19][9]/Q (SEL_FDPRBQ_D_1P5)	0.10	0.10 r
mult_69_G20/B[9] (filter_DW02_mult_19)	0.00	0.10 r
...		
mult_69_G20/PRODUCT[23] (filter_DW02_mult_19)	0.00	1.95 f
add_15_root_add_0_root_add_80/B[23] (filter_DW01_add_107)	0.00	1.95 f
...		
add_15_root_add_0_root_add_80/SUM[24] (filter_DW01_add_107)	0.00	2.21 r
add_9_root_add_0_root_add_80/B[24] (filter_DW01_add_64)	0.00	2.21 r
...		
add_9_root_add_0_root_add_80/SUM[26] (filter_DW01_add_64)	0.00	2.43 f
add_4_root_add_0_root_add_80/B[26] (filter_DW01_add_203)	0.00	2.43 f
...		
add_4_root_add_0_root_add_80/SUM[29] (filter_DW01_add_203)	0.00	2.70 f
add_1_root_add_0_root_add_80/A[29] (filter_DW01_add_50)	0.00	2.70 f
...		
add_1_root_add_0_root_add_80/SUM[31] (filter_DW01_add_50)	0.00	2.94 f
add_0_root_add_0_root_add_80/B[31] (filter_DW01_add_52)	0.00	2.94 f
...		
add_0_root_add_0_root_add_80/SUM[31] (filter_DW01_add_52)	0.00	3.03 f
y_out_reg[15]/D (SEL_FDPRBQ_D_1P5)	0.00	3.03 f
data arrival time		3.03

Hyeong-Ju Kang

63



Pipeline



- ▶ clock frequency – unpipelined
 - ▶ path delay 2.93ns + other 0.14ns = 3.07ns
 - ▶ → 326MHz
- ▶ expected clock frequency - pipelined
 - ▶ path1 delay 1.85ns + other 0.14ns = 1.99ns
 - ▶ path2 delay 1.08ns + other 0.14ns = 1.22ns
 - ▶ → 503MHz

Hyeong-Ju Kang

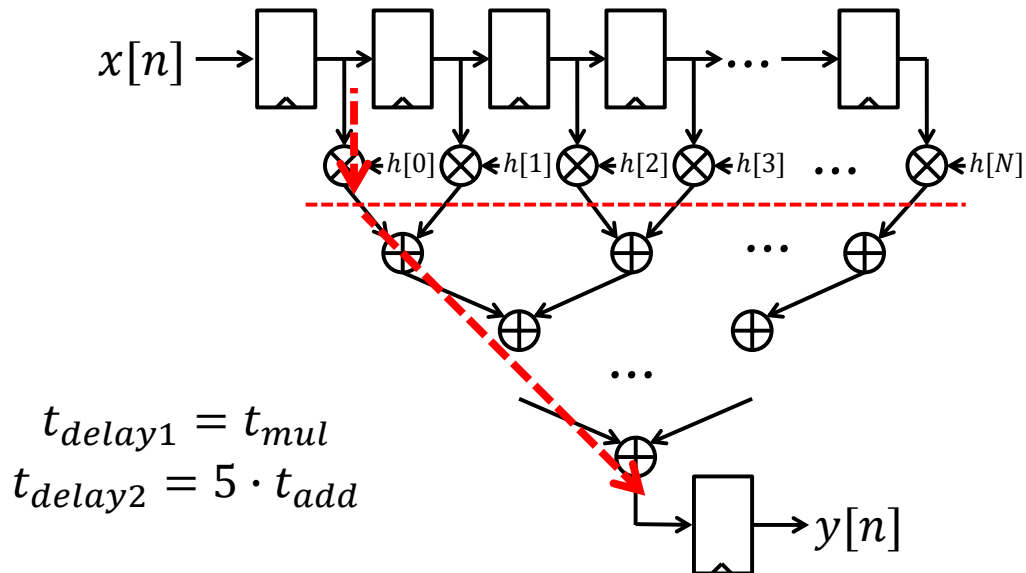
64



Critical Path - Pipeline



- ▶ 예상 critical path – adder tree & pipeline



Pipeline in Verilog



```

genvar i;
wire signed [31:0] mul[20:0];
for(i=0;i<21;i++) begin
    assign mul[i] = x[i] * h[i];
end

```



```

reg signed [31:0] mul[20:0];
always@(posedge clk or negedge n_reset)
begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<21;i++) begin
            mul[i] <= 'b0;
        end
    end else begin
        for(int i=0;i<21;i++) begin
            mul[i] <= x[i] * h[i];
        end
    end
end
end

```

○○○ 합성 결과 – Critical Path Delay ○○○

Point	Incr	Path
mul_reg[1][2]/CK (SEL_FDPRBQ_D_1P5)	0.00 #	0.00 r
mul_reg[1][2]/Q (SEL_FDPRBQ_D_1P5)	0.09	0.09 f
add_16_root_add_0_root_add_88/A[2] (filter_DW01_add_132)	0.00	0.09 f
...		
add_16_root_add_0_root_add_88/SUM[3] (filter_DW01_add_132)	0.00	0.30 r
add_7_root_add_0_root_add_88/B[3] (filter_DW01_add_131)	0.00	0.30 r
...		
add_7_root_add_0_root_add_88/SUM[8] (filter_DW01_add_131)	0.00	0.67 f
add_2_root_add_0_root_add_88/B[8] (filter_DW01_add_127)	0.00	0.67 f
...		
add_2_root_add_0_root_add_88/SUM[10] (filter_DW01_add_127)	0.00	0.98 r
add_1_root_add_0_root_add_88/B[10] (filter_DW01_add_165)	0.00	0.98 r
...		
add_1_root_add_0_root_add_88/SUM[24] (filter_DW01_add_165)	0.00	1.53 f
add_0_root_add_0_root_add_88/B[24] (filter_DW01_add_137)	0.00	1.53 f
...		
add_0_root_add_0_root_add_88/SUM[31] (filter_DW01_add_137)	0.00	1.80 r
y_out_reg[15]/D (SEL_FDPRBQ_D_1P5)	0.00	1.80 r
data arrival time		1.80

0.21ns

0.37ns

0.31ns

0.55ns

0.27ns

Hyeong-Ju Kang

67

○○○ Pipeline ○○○

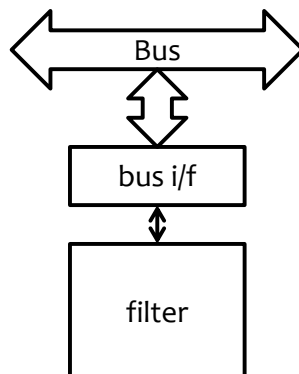
- ▶ clock frequency – unpipelined
 - ▶ path delay 2.93ns + other 0.14ns = 3.07ns
 - ▶ → 326MHz
- ▶ clock frequency – pipelined
 - ▶ expected
 - ▶ path1 delay 1.85ns + other 0.14ns = 1.99ns
 - ▶ path2 delay 1.08ns + other 0.14ns = 1.22ns
 - ▶ → 503MHz
 - ▶ in real
 - ▶ path2 delay 1.71ns + other 0.13ns = 1.84ns
 - ▶ → 543MHz

Hyeong-Ju Kang

68

- ▶ Convolution
- ▶ Filter의 설계 및 C 언어 구현
- ▶ Fixed Point 구현
- ▶ Verilog 구현
- ▶ 합성과 pipeline
- ▶ Reusable Design

- ▶ 외부에서 버스 등을 통해 계수 지정



```

module filter (
    input          clk,
    input          n_reset,
    input [15:0]   x_in,
    output reg [15:0] y_out,

    input          h_write,
    input [4:0]    h_idx,
    input [15:0]   h_data
);
...
reg signed [15:0] h[20:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<21;i++) begin
            h[i] <= 'b0;
        end
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end
end

```

coefficient write signals

reset

write

→ Testbench도 수정해야 함

end

Hyeong-Ju Kang

71

- ▶ Coding style
 - ▶ 되도록 모든 의미 있는 상수에는 이름을
 - ▶ C언어에서는 매크로 정의(#define) 사용
- ▶ Verilog에서 상수
 - ▶ `define
 - ▶ resolved at compile time
 - ▶ parameter, localparam
 - ▶ resolved at elaboration time



Parameter



```
module filter #(
    NUM_TAP = 21
) (
    input clk,
    input n_reset,
    ...
);

reg signed [15:0] h[NUM_TAP-1:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0; i<NUM_TAP; i++) begin
            x[i] <= 'b0;
        end
    end else begin
        x[0] <= x_in;
        for(int i=1; i<NUM_TAP; i++) begin
            x[i] <= x[i-1];
        end
    end
end
end
```

NUM_TAP = 21 → Tap 개수를 parameter로 지정

NUM_TAP 파라미터 이용



Instantiation



- ▶ 다양한 길이의 filter를 하나의 코드로 만들 수 있음

```
filter #(NUM_TAP(21)) i_filter21 (.clk(clk), .n_reset(n_reset), ...);
filter #(NUM_TAP(51)) i_filter51 (.clk(clk), .n_reset(n_reset), ...);
```

- ▶ Resolution time

- ▶ `define: resolved at compile time
 - ▶ Verilog code를 읽어 들일 때 상수값이 고정됨
 - ▶ Instance 마다 서로 다른 값을 사용할 수 없음
- ▶ parameter: resolved at elaboration time
 - ▶ Instantiation을 할 때 상수값들이 결정됨
 - ▶ Instance 마다 서로 다른 값을 사용할 수 있음

- ▶ Width parameters
 - ▶ Input data: total width, fractional part
 - ▶ Coefficients: total width, fractional part
 - ▶ Output data: total width, fractional part
 - ▶ Other widths should be calculated from these parameters.

```

localparam W_MUL = W_ID + W_C - 1;
localparam F_MUL = F_ID + F_C;
localparam W_SUM = W_MUL + $clog2(NUM_TAP);

wire signed [W_MUL-1:0] mul[20:0];
wire signed [W_SUM-1:0] sum;
...
wire signed [W_SUM-1:0] y_rnd = sum + (1<<(F_MUL-F_OD-1));

always@(posedge clk or negedge n_reset) begin
  if(n_reset == 1'b0) begin
    y_out <= 'b0;
  end else begin
    y_out <= y_rnd[F_MUL-F_OD+W_OD:F_MUL-F_OD];
  end
end
endmodule

```

saturation 처리도 필요?