Department
of
Electrical and Information Technology

DSP-Design:

Laboratory Manual Lab 3

Lab 3

Effects of fixed-point implementation in DSP algorithms

This session deals with effects that arise when mapping a DSP algorithm onto a fixed-point computation platform. In a hardware design flow one usually creates such a fixed-point model and compares it against a floating-point model in order to estimate the complexity and computation effort of datapaths for a desired performance. As an example, the performance of a *Finite Impulse Response* (FIR) filter will be characterized and estimated in terms of its wordlength, area, and speed. Since this session will be done in MATLAB, some important commands appear on the margin of a page. **You can get more information on these and other functions by typing `help` and the respective command name in the MATLAB command window**. cmr

`command name`

In order to pass the lab sessions, you have to accomplish the listed compulsory assignments before the respective session

It is also highly recommended to try to solve the programming parts beforehand since this improves the use of the lab. This manual also tries to give you some introduction to the basic commands and techniques to be used.

1.1 FIR filters

Based on difference equations, the output of a discrete *Linear Time-Invariant* (LTI) system in the time domain can be described as

$$y(n) = \sum_{i=0}^{m} b_i x(n-i) - \sum_{j=1}^{l} a_j y(n-j). \tag{1.1}$$

Such a system is called *Infinite Impulse Response* (IIR) filter. Removing the feedback part of this system by setting $l = 0$, it follows that the current output only depends on the current input and its delayed samples. However, in order to achieve the same performance, such a filter needs a higher order compared to the system

described by (1.1). Note that a higher filter order results in a larger delay.

$$y(n) = \sum_{i=0}^{m} b_i x(n-i) \equiv \sum_{i=0}^{m} h_i x(n-i) \tag{1.2}$$

The system is of order m and non-recursive. Its impulse response $\mathbf{h} = \{h_0, \ldots, h_m\}$ is limited to $m+1$ taps and systems of this type are called FIR filter.

FIR filters are always stable and they can be designed to have linear phase, that is, the group delay time is constant. Then, the phase of an input signal will not be distorted when passing through the filter. A block diagram of (1.2) in direct form is shown in Figure 1.1.

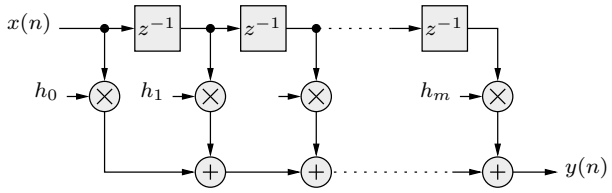

Figure 1.1: FIR filter in direct form.

Doing a graph transposition on the preceding architecture, that is, reversing the direction of all edges, exchanging the input and output nodes while keeping the edge delay the same, yields another realization of an FIR filter, namely the transposed or data-broadcast structure (Figure 1.2).

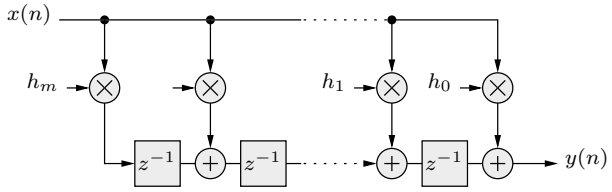

Figure 1.2: FIR filter in transposed form, data-broadcast.

When it comes to implementation in hardware, area and speed are important constraint parameters. In this case, area primarily depends on the wordlength that has to be used for the different arithmetic units to maintain accuracy. Speed can be measured in terms of the longest path between any two delay elements, called critical path.

1.2 Real-life considerations

1.2.1 Calculating the transfer function

In order to understand the following considerations about transfer functions, we will need

Definition 1 *Discrete Fourier Transform (DFT)*

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} = \mathrm{DFT}\{x(n)\} \tag{1.3}$$

The transform $\mathbf{X}$ *consists of* N *elements, that is,* $\mathbf{X} = \{X(0), \ldots, X(N-1)\}$.

Generally, transfer function **H** and impulse response **h** are two ways of describing an LTI-system. The first description is used in the frequency domain, the latter in the time domain. Together with the preceding definition, it is seen that these two descriptions are connected for **discrete-time** systems by the DFT:

$$\mathrm{DFT}\{h_n\} = H(k).$$

As mentioned in the introduction, implementing an algorithm on a hardware platform will always result in performance degradation due to finite wordlength in the computation units. Generally, an analog input signal has to be sampled and quantized before it can be processed by a digital circuit. For our example we assume that we receive samples in 2's complement coded with w_{in} bits.

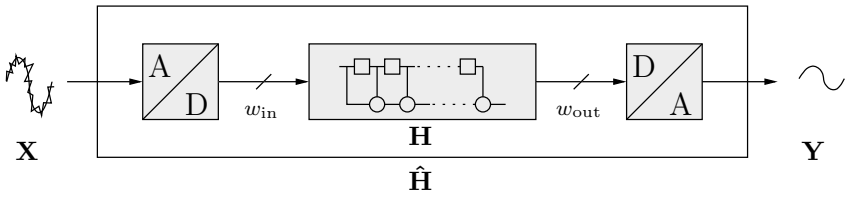

Figure 1.3: Block diagram of a common DSP system.

Seen from a system's perspective, introducing nonlinear elements, such as A/D- and D/A-converters, makes the original linear system with transfer function **H** into nonlinear system $\hat{\mathbf{H}}$ (Figure 1.3). Hence, $\hat{\mathbf{H}}$ can only be an estimation of the expected system transfer function since it depends on the respective input stream and the precision used for the converters. Due to absence of linearity,

$$Y(k) \neq H(k)X(k).$$

However, given a certain **X**, w_{in}, and w_{out}, this configuration resembles one certain realization $\hat{\mathbf{H}}$ of the transfer function **H**. This realization is linear itself and we can assume that

$$Y(k) = \hat{H}(k)X(k)$$

holds.

Now, let $x(n)$ and h_n be finite sequences with length L and M, respectively. Then, the output sequence $y(n)$ of an FIR filter is defined by (1.2), where $m = M-1$, and its length is $L + M - 1$. This operation corresponds to a convolution of **x** with **h**. `conv`

The frequency domain equivalent to (1.2) is

$$Y(\omega) = H(\omega)X(\omega).$$

Notice that this spectrum is continuous. In order to represent $y(n)$ uniquely in the frequency domain at a set of discrete frequencies, the number of DFT samples N must be at least $N \geq L + M - 1$. Hence,

$$\begin{aligned} Y(k) &= Y(\omega)\big|_{\omega=2\pi k/N} = H(\omega)X(\omega)\big|_{\omega=2\pi k/N} \\ &= H(k)X(k). \end{aligned}$$

Note that the sequences h_n must be padded with zeros to increase its length to N in order to calculate the transfer function $\hat{\mathbf{H}}$.

$$\hat{H}(k) = \frac{Y(k)}{X(k)}$$

If the number of frequency points is a power of 2, (1.3) can be carried out as *Fast Fourier Transformation* (FFT), which is more computation efficient. Otherwise the sequences can always be padded with zeros to achieve this requirement. `fft`

1.2.2 Errors due to finite wordlength arithmetic

The real frequency response of an FIR filter differs from the ideal one due to:

- Quantization of the filter coefficients ⇒ linear error
- Truncation of partial products ⇒ nonlinear, random error

Algebraically, (1.2) can be seen as multiplication of two vectors $\mathbf{h}$ and $\mathbf{x}$, and according to the first consideration, the impulse response $\mathbf{h}$ can now be written as $\mathbf{h}_q$, with $\mathbf{h}_q$ being the quantized impulse response. Hence, the error due to coefficient quantization is

$$\mathbf{\Delta y} = \mathbf{h} \cdot \mathbf{x}^T - \mathbf{h}_q \cdot \mathbf{x}^T = \sum_{i=0}^{m} (h_i - h_{i,q}) \cdot x_i = \sum_{i=0}^{m} \Delta h_i \cdot x_i.$$

This error is linearly dependent on the input stream and thus causes only linear distortion. The filter can therefore be modeled as depicted in Figure 1.4.

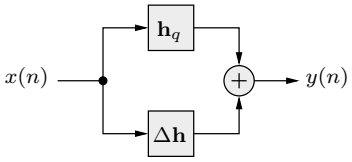

Figure 1.4: Model of an FIR filter with quantized coefficients.

The second error source depends on how the wordlength of partial products is adjusted when maximum accuracy cannot be kept throughout processing. One can either truncate or round a number to maintain a certain wordlength. Obviously, truncating a number in hardware is easier than rounding, where additional logic has to perform a comparison. However, even if you keep the required accuracy in your calculations, you have to truncate the result before the following D/A-converter, since this stage can only handle a certain precision w_{out}, usually between 8 and 16 bits.

`round` `floor`

Notice that these two errors can be handled quite easily in non-recursive systems since they are stable by definition. In recursive systems, however, instable behavior can be introduced by improper quantization and/or scaling approaches, see Section 1.4.

How do you truncate at the output?

The calculation in assignment ④ now comes into play. The expected result has got a wordlength that differs from the theoretical one (w) that considers the worst case where all taps of the impulse response are 1. Since the impulse response is usually known in advance, clipping can be done according to Figure 1.5.

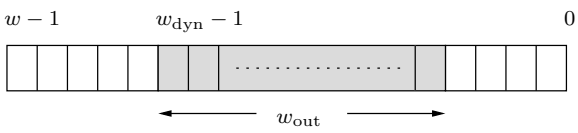

Figure 1.5: Dynamic range of the output result.

This improves the performance compared to a truncation starting from $w-1$ because there is no information in the upper range; the result has only a certain

dynamic range determined by w_{dyn}, where

$$w_{\text{dyn}}(\mathbf{h}) \leq w.$$

However, this operation certainly removes information from the signal (the *Least Significant Bits* (LSBs) are lost!) and therefore this system will behave somewhat different compared to an ideal implementation.

Compulsory assignments

① Given an FIR filter order m, compare the presented architectures with regard to area and speed according to the given remarks.

- Which wordlength is required in the adders, multipliers, and registers in these approaches if full precision is required? Assume that the wordlength of the input and the coefficients are w_{in} and w_{c}, respectively.
- Determine the critical path computation time T_{crit} as a function of T_{add} and T_{mult}.

② Find and draw another architecture that overcomes to some extent the main disadvantage of the direct form. Compare it to the previous designs. Where is the improvement?

③ Can you think of another solution to improve the direct form? What is the drawback?

④ Given an impulse response $\mathbf{h}$, express the required dynamic wordlength w_{dyn} at the output as function of h_i, w_{in}, and m. Assume that numbers in this implementation are represented as integers in 2's complement[1].

⑤ The impulse response of a linear-phase FIR filter has a special property. Which one? Improve the architectures in Figure 1.1 and 1.2 by making use of this property.

⑥ You should be able to explain the working of the following commands

- `floor`
- `round`
- `fir1`
- `fft`
- `abs`
- `plot`
- `angle`
- `conv`

You may use `help` *`command_name`* in the MATLAB prompt to know more about each command.

⑦ In this assignment you should write some initial matlab code. **The code should have been tested before the lab session.** Design a lowpass filter with a cutoff frequency of 0.1 and using the `fir1` command and plot the impulse response `h`. Show the filter's frequency response, both magnitude (in dB) and phase. Use a filter order of your choice.

The transfer function can be obtained by applying an FFT to the filter coefficients (command in MATLAB: `fft`). The magnitude of the transfer function can be calculated as

[1] A number in 2's complement covers the range $[-2^{w-1}, 2^{w-1} - 1]$.

```
20*log10(abs(H)+(H==0)*eps);
```

where H is the output of the FFT and (`H == 0`) ∗ `eps` suppresses warnings that would result from taking the logarithm of zero. The phase response is calculated using

```
unwrap(angle(H));
```

1.3 During the lab

Start **MATLAB** in a new directory where you write the script files for the lab. Throughout the lab, use these script files for execution and displaying the outputs.

1.3.1 Algorithm simulation

When designing a (digital) filter you have to know some parameters beforehand. According to the sampling theorem, the sampling frequency f_s of a system has to be at least two times the maximum signal frequency. Hence, possible filter bands lie within $0 < f < f_s/2$. Normalizing a desired cutoff frequency f_c to the sampling rate yields $0 < f_c < 1/2$, with $f_c = f/f_s$. Another parameter that mainly determines the performance of the filter is its order m. As a rule of thumb, the higher the order, the better the performance but the more expense in computation.

1.3.2 Lab assignments

The lab session guides you through the following steps (Figure 1.6) that are crucial in designing a digital filter and estimating performance losses upon implementation in hardware.

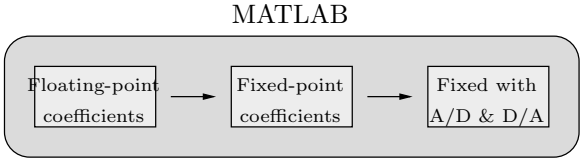

Figure 1.6: Steps in the lab session.

Start with the floating point model that MATLAB provides. Desired filter specifications are set and real-numbered coefficients are obtained. Then, the coefficients are quantized, and finally, effects from input and output signal quantization are simulated.

`fir1` `fft` `angle` `unwrap`

❶ Design a lowpass filter with a cutoff frequency of 0.1 and filter order $m \in \{7, 15, 31\}$ and plot their magnitude (in dB) and phase responses . When you use filters of different orders m, what can you say about the performance?

❷ Now, quantize the real valued coefficients of the filter from the previous assignment to w_c bits and plot the respective transfer functions. Try different w_c to see the effects of different precisions. In order to fully exploit the dynamic number range, it is advisable to normalize the impulse response to avoid the chances of overflow. The filter coefficients obtained from `fir1` is already scaled so the center of the first pass band has magnitude exactly one after windowing. When you plotted the transfer functions, elaborate on what you see.

Normalization of a vector x with respect to its maximum absolute value is done by

```
x/max(abs(x));
```

Quantization of the coefficients to w bits is done by a multiplication, followed by a rounding, and a division.

```
y=x*2^(w-1); z=round(y); x=z/2^(w-1);
```

❸ We will now try to verify the model in Figure 1.4. If we account for the error introduced by quantization, the output should be same as the one when calculated with full precision. To check this you have to show the difference between the two outputs. $\mathbf{y} = \mathbf{h} * \mathbf{x}$ is the output with full precision. Here, $*$ denotes a convolution! $\mathbf{y}_s = \mathbf{h_q} * \mathbf{x} + \mathbf{\Delta h} * \mathbf{x}$ will be the output as calculated from Figure 1.4, that is, taking into account the error introduced during the quantization process. What should the difference between the outputs $(\mathbf{y} - \mathbf{y}_s)$ be? Does the simulation confirm your assumption? `conv`

The input stream is provided in `input.dat` and has to be read into the variable $\mathbf{x}$ before calculating $\mathbf{y}$ and $\mathbf{y}_s$. Loading the variable can be done with

```
x=dlmread('input.dat', ' ');
```

What has been seen are the linear errors due to coefficient quantization. However, what is left to do is to look at effects that arise from the truncation of computation products. Furthermore, the input samples are also wordlength-limited and have to be taken into account in the overall model as well.

❹ Do an A/D-conversion of $\mathbf{x}$ with $w_{\text{in}} = 8$ bits.

```
x_temp=x*2^(w-1); x_temp2=floor(x_temp); x_q = x_temp2/2^w-1
```

Filter this signal with the quantized ($w_{\text{c}} = 8$) impulse response (h_q) of the previous filter ($m = 31$). Then truncate the result to the output wordlength i.e., $w_{\text{out}} = 9$. In other words, you have to convolve x_q with the quantized impulse response h_q.

```
y_q = conv(h_q,x_q);
```

❺ Now, plot the transfer function $H = Y./X$, where Y is the `fft` of the truncated output stream (y_q from ❹) and X the `fft` of the input stream from `input.dat`. What is different compared to the original transfer function that was obtained in ❶. *Note that transfer functions of same filter order are to be compared*

1.4 Finite wordlength issues in IIR filters

In this section you will simulate the effects of a parasitic oscillation, namely the zero-input oscillation, which is due to finite wordlengths in IIR filters.

In an IIR filter the data wordlength increases when a signal is multiplied by a coefficient and would therefore become infinite in a recursive loop. Hence, quantization, that is, truncation or rounding to the original wordlength, is necessary. This quantization is a nonlinear operation in recursive algorithms.

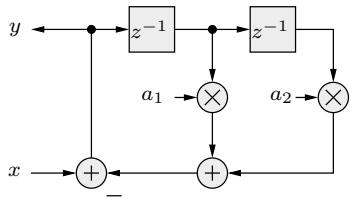

Figure 1.7: Direct form structure of an IIR filter.

❼ Apply the sinusoidal signal

$$x(t) = \begin{cases} -0.1 \cdot \sin(2\pi \cdot t), & 0 \leq t \leq 1.25 \\ 0, & \text{otherwise} \end{cases}$$

to the direct form IIR filter depicted in Figure 1.7. Filter this signal with the tap set $b = 1, a = [a_0, a_1, a_2] = [1, -489/256, 15/16]$ and using the `filter` function. Plot both the input and output stream in one picture.

❽ Now, demonstrate by simulation that zero-input oscillations occur when you quantize the outputs of the two multipliers of the filter in Figure 1.7 to 6 bits (2's complement). Try both with truncation and rounding and plot the two resulting output streams in one figure together with the input stream. **HINT: In order to quantize the multiplier outputs the entire filter has to be expressed as an equation.**

- What is the amplitude of the oscillation in the rounded version in terms of the quantization level $Q = 1/2^5$?
- What is the DC offset in the truncated version?
- Observe what happens to the output when you have different step sizes while constructing the signal $x(t)$. For example, if the step size is 0.01 and 0.001. What could be the reason?