Digital Filter Implementation 2

442.003 Digital Signal Processing, Laboratory
Winter Term 2020/21

Signal Processing and Speech Communication Laboratory
www.spsc.tugraz.at

Last updated: October 28, 2020

Abstract

In the first experiment of this laboratory, we examine a source code example of a cascaded biquad IIR filter implemented in fixed-point arithmetic. The second task deals with zero-input limit cycles in fixed-point IIR filters and in the last experiment, we use an FIR Hilbert transformer to implement a single-sideband modulator.

1 Theoretical Overview

1.1 Limit Cycles

A limit cycle[1] is an isolated closed trajectory in the state space[2] of a system (see [1]). Whenever the system's state advances on a closed trajectory, the system exhibits oscillations. Limit cycles are inherently nonlinear phenomena—they can't occur in linear systems. Of course, a linear system of at least second order can produce oscillations (e.g. a linear filter with a pole on the unit circle), but their closed trajectories are not isolated (neighboring trajectories are closed too—the amplitude of the oscillation of a linear system is set entirely by the initial conditions).

When we implement an IIR filter (feedback loops) with finite-precision arithmetic, we will get a nonlinear dynamic system and therefore oscillations may occur even when the (quantized) filter coefficients yield a stable (linear) filter. We distinguish between

1. limit cycles due to overflow (modulo behavior) after accumulation and
2. limit cycles due to quantization (truncation or rounding) after multiplication or accumulation.

The first produces oscillations with high amplitudes, whereas the second exhibits oscillations with low amplitudes which are disturbing especially if no input signal is applied to the filter (zero input). Fortunately, limit cycles of the first type can be avoided in second-order systems by using a saturation characteristic. For more information refer to [2].

[1] In German: Grenzzyklus
[2] Q: What is the state space of a digital filter?

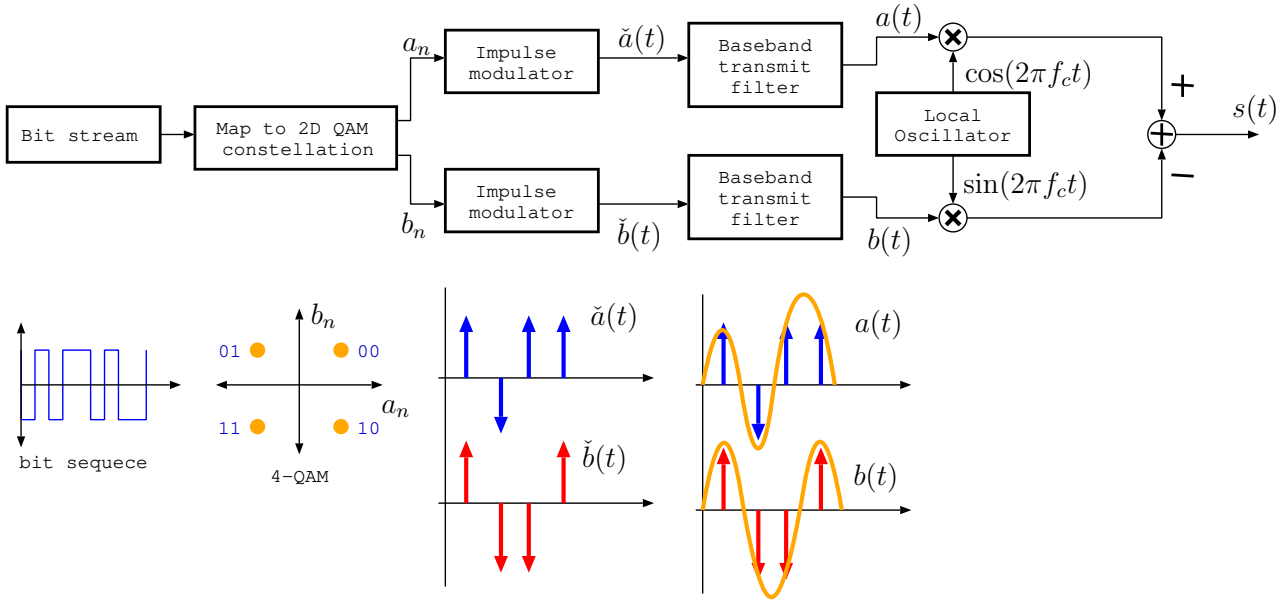

Figure 1: A Basic QAM modulator.

1.2 QAM modulator

Quadrature Amplitude modulation (QAM) is a widely used digital modulation technique. The goal of the digital modulation is to construct the baseband representation of the modulated digital signal, which is then modulated by the carrier frequency to the proper frequency band. It is often used as a modulation scheme in standard telephone modems, as well as in FAX modems, wireless LAN standards and many other digital communication systems. In this laboratory we will only consider a QAM modulator.

The block/diagram of the QAM modulator is shown in Figure 1. The input data bits d that arrive at the rate of $F_{data} = 1/T_{data}$ bps are converted into J-bit binary words by simply concatenating consecutive bits. Each J-bit word selects a channel symbol from the 2^J alphabet, given by the corresponding QAM constellation Fig.2. Each symbol in the alphabet is represented by a corresponding complex number $c_n = a_n + jb_n$ selected from the QAM constellation:

Word	Signal point
'00'	$1 + j1$
'01'	$-1 + j1$
'10'	$1 - j1$
'11'	$-1 - j1$

This mapping will correspond to the symbol rate (also known as baud rate) of $F_{symb} = F_{data}/J$. It is customary to call the real part of the symbol, a_n, the *in-phase* component or I component, and the imaginary part, b_n, the quadrature or Q component. There are several QAM constellation sizes used in practice: $J = 2$ (4-QAM), $J = 4$ (16-QAM), $J = 6$ (64-QAM), $J = 8$ (256-QAM), and $J = 10$ (1024-QAM) (see examples in Fig. 2).

Case $J = 10$ provides the highest bit rate for the same baud rate, however this regime requires very high SNR in order to achieve acceptable bit-error rate.

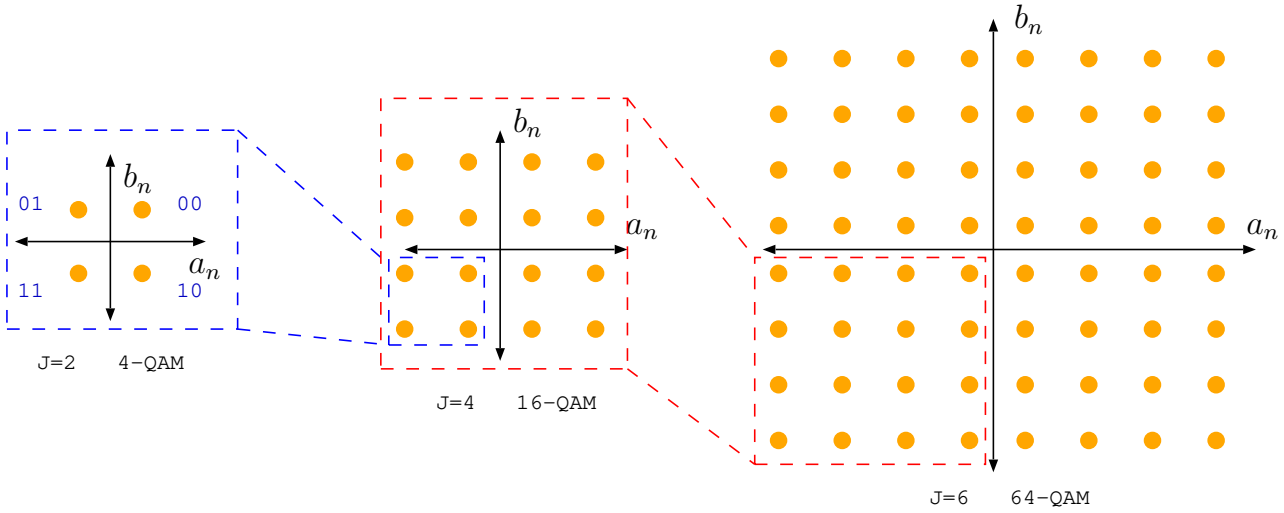

Figure 2: Examples of the mappings for 4-QAM, 16-QAM, and 64-QAM constellations.

The I and Q components are then passed through the impulse modulators

$$\check{a}(t) = \sum_{k=-\infty}^{\infty} a_k \delta(t - kT_{symb}),$$

and

$$\check{b}(t) = \sum_{k=-\infty}^{\infty} b_k \delta(t - kT_{symb}),$$

resulting in the pulse-modulated representation of the transmitted symbols. These signals are then passed through the identical baseband transmit shaping filters, each with the impulse response $g(t)$. The baseband transmit filter is typically a lowpass filter approximating the raised cosine or square-root of cosine response, so its cutoff frequency is somewhat greater than $F_{symb}/2$. The role of the shaping filters is to restrict the bandwidth of the transmitted signal. The output of the transmit filters can be then represented as

$$a(t) = \sum_{k=-\infty}^{\infty} a_k g(t - kT_{symb}),$$

$$b(t) = \sum_{k=-\infty}^{\infty} b_k g(t - kT_{symb}).$$

In order to translate the baseband modulated signal into the proper passband, in-phase and quadrature components $a(t)$ and $b(t)$ are *double-sideband suppressed-carrier amplitude modulated* by the carriers $\cos(2\pi f_c t)$ and $\sin(2\pi f_c t)$ as shown in Fig. 1. The resulting QAM signal $s(t)$ is then formed as

$$s(t) = a(t)\cos(2\pi f_c t) - b(t)\sin(2\pi f_c t)$$

The receiver can then recover the quadrature and in-phase components from $s(t)$ by exploiting the Hilbert transform of $s(t)$ (for more information see [3]):

$$\tilde{s}(t) = \Big[s(t) + \mathcal{H}\{s(t)\}\Big] \mathrm{e}^{-j2\pi f_c} = \sum_{k=-\infty}^{\infty} (a_k + jb_k) g(t - kT_{symb}),$$

where $\mathcal{H}\{\cdot\}$ is a Hilbert Transform operator. The demodulated signal $\tilde{s}(t)$ is then sampled at the symbol period $\tilde{s}(nT_{symb}) = (a_k + jb_k)$ and the corresponding values are mapped back into the bit sequence.

2 Practical Part

Experiment 1:
Fixed-Point Implementation of a Cascade Form IIR Filter

Equipment: Your brain
Software: Knowledge

1. Examine the C code in Figure 4. This is the C equivalent of the hand-optimized assembly routine taken from an Texas Instruments DSP. Draw the signal-flow graph that is implemented by the code within the for loop.

2. Answer the following questions:

   (a) Which numerical format (number of bits for the integer portion and for the fractional portion, significance of bits) is expected for the filter coefficients?

   (b) Which numerical range is resulted for the filter coefficients?

   (c) Draw the area in the z-plane where poles (or zeros) can be located. Is it possible to use this routine for a general IIR filter?

   (d) Which numerical format should be used when we want to be able to place poles anywhere within the unit circle?

```
void iir_cas4( int n_cas, short *coeffs, int *states, int *io)
{ int k0, k1, i;
  for( i = 0; i < n_cas; i++)
  { k0    = coeffs[4*i+1] * (states[2*i+1] >> 16) +
            coeffs[4*i+0] * (states[2*i+0] >> 16) + io[0];
    io[0] = coeffs[4*i+3] * (states[2*i+1] >> 16) +
            coeffs[4*i+2] * (states[2*i+0] >> 16) + k0;
    states[2*i+1] = k0;
    k1    = coeffs[4*i+1] * (states[2*i+0] >> 16) +
            coeffs[4*i+0] * (k0 >> 16) + io[1];
    io[1] = coeffs[4*i+3] * (states[2*i+0] >> 16) +
            coeffs[4*i+2] * (k0 >> 16) + k1;
    states[2*i+0] = k1;
  }
}
```

Figure 3: *C equivalent of an IIR filter.*

Experiment 2:

Limit Cycle due to ?

Equipment: PC + RPi, headphones

Software: netbeans, download `/courses/dsplab/filt2` and unzip it on your workstation

1. Plug the output of the PC soundcard (or a CD player) to the RPi input and the headphones to the RPi output.

2. In netbeans, load the provided project file `Exp2`.

3. Edit the filter coefficients in `iir2.cpp` according to the following table ($Q15 = 2^{15}$ is already defined).

b(0)	b(1)	b(2)	a(0)	a(1)	a(2)
Q15	0	0	Q15	-Q15*3/4	Q15*3/4

4. Build the program, load it to the RPi, and run it. Provide an input signal to the RPi and listen to the output. If you can hear a non-distorted output signal we can assume the filter works properly.

5. Edit the file `iir2.cpp` again: set the initial conditions to

y(-2)	y(-1)
-Q15*4/5	Q15*4/5

6. Rebuild the program and load it to the RPi. **Before** you run the program, ensure the amplitude of the provided input signal is **zero** (e.g., use the soundcard's mixer to set the volume) and ensure you **do not wear the headphones**. What can you observe at the RPi's output?

7. Continuously increase the amplitude (volume) of the provided input signal. Does this change the program's behavior?

8. In order to see more clearly what's going on, print the output samples `in` of each frame using `for(int i=0;i<framelength;i++){ std::cout<<out[i]<<std::endl;}` Set the filter coefficient `b(0)=0` in `iir2.cpp` to force zero input. Rebuild, reload, and run the program. It is not necessary to provide an input signal to the RPi now, because all feed-forward coefficients are zero.

   Answer whether these plotted samples can be the zero-input response of a stable linear system or not (explain). Analyze the source code in `iir2.cpp` and find out whether there is a nonlinearity that causes the observed behavior.

Experiment 3:
QAM modulator

1. Connect left and right RPi channels to the oscilloscope.

2. If necessary start the netbeans and load the `Exp3` project. Open the `Quam.h` file and set `MODULATE 0`, `SINGLECHANNEL 0`, and `INC_TRANS_CHANNEL 0`. This will disable the carrier modulation, and will result in the in-phase and quadrature components outputted separately over the left and right channels. The last definition will switch off the simulation of the communication channel influence.

3. Now, let us start with the 4-QAM scheme. Set `#define QAM 4`, build the program and run it.

4. By default the program will only perform simple impulse modulation without the usage of transmit filters. Using the `fir()` function, which implements an FIR filtering, and raised cosine impulse response, stored in the variable `G[]` implement the corresponding transmit filter. Your code should go in the `qam()` method in `Quam.cpp`.

5. On the oscilloscope horizontal control panel press **Main/Delayed** button and select ***XY*** mode. If the modulation is disabled, you should be able to see the selected QAM constellation. Keep in mind, that left and right channels are used to output real and imaginary parts of the symbols we are transmitting. You can also enable averaging to remove some noise.

6. Now, try including the channel. First, set `INC_TRANS_CHANNEL 1`. This will simulate a simple communication channel with an impulse response $h[n] = (c + jd)\delta[n]$. Note how the channel influence the signal constellation. In the `Quam.cpp` file find method `channel()` that simulates the channel and try setting other values for the channel coefficients `c` and `d`.

7. By setting `INC_TRANS_CHANNEL 2` you can simulate the influence of the time-varying channel with the impulse response $h_t[n] = (c(t) + jd(t))\delta[n]$. Note that the signal constellation is now changes with time, thus requiring an adaptive equalizer, that cancels the time-varying nature of the channel.

8. Now, let us return to the normal display mode by pressing **Main/Delayed**→**Main**.

9. In the source code set `SINGLECHANNEL 1`. In this mode one of the channels will be used to transmit the sync signal that changes its state at the symbol boundary, while the other channel is difference between the in-phase and the quadrature components. The latter is real valued and it is exactly the signal that is then transmitted over the channel.

10. Then, press **Math**→***FFT***→***Setting*** and set ***Source*** to the channel that corresponds to the information signal, ***Span*** to 20kHz, and ***Center*** to 10kHz.

11. Measure the zero-to-zero signal bandwidth B as given by the FFT spectrum.

12. For the case when each symbol consist of `N` samples, and sampling rate F_s is set to $F_s = 32$kHz, compute the symbol period T_{symb} and the corresponding baud rate F_{symb} (symbols/sec) for this system. What is then the corresponding data rate F_{data} (bps)? Compare these numbers to the bandwidth of the transmitted signal.

13. Now, enable the modulation by setting `MODULATE 1` in the source code.

14. Measure the center frequency F_c and the bandwidth of the resulting channel.

15. Try other modulation schemes, i.e., 16-QAM, 64-QAM, 256-QAM, and 1024-QAM. What is the bandwidth of the modulated signal in these cases? What are the corresponding baud and bits rates in these cases?

A Credits

This document was authored and/or adapted by Christian Feldbauer and Josef Kulmer.

References

[1] Strogatz, S.H., "Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering," Addison-Wesley, Reading, MA, 1994.

[2] Oppenheim, A.V. and Schafer, R.W.: "Discrete-Time Signal Processing," Second Edition, Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1999.

[3] Doblinger, G.: "Signalprozessoren. Architekturen—Algorithmen—Anwendungen," J. Schlembach Fachverlag, Weil der Stadt, Deutschland, 2000.

[3] Bernard Sklar, "Digital Communication - Fundamentals and Applications," Prentice Hall PTR, 2000.