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.

1 Theoretical Overview

1.1 Limit Cycles

A limit cycle\(^1\) is an isolated closed trajectory in the state-space of a system (see [?]). 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 [?].

\(^1\)ger: Grenzzyklus
1.2 Data Types and Word Lengths

Make sure that you are familiar with data types to read assembly code. The C/C++ data types and word lengths are as follows:

- **bool**: 8 bits (top 7 bits ignored)
- **char**: 8 bits
- **short**: 16 bits
- **int**: 32 bits
- **long**: 32 to 64 bits
- **long long**: 64 bits
- **float**: 32 bits
- **double**: 64 bits
- **long double**: 96 bits
- **pointer**: 32 bits

1.3 Assembly Routine \texttt{iir\_cas4()}\texttt{( )}

You will need the code of the following assembly routine in experiment 1: fixed-point implementation of a cascade form IIR filter. Before attending this laboratory unit, make sure that you are familiar with right-shifts, numerical formats, and biquad filters.

```c
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];
                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];
                coeffs[4*i+2] * (k0 >> 16) + k1;
        states[2*i+0] = k1;
    }
}
```

Figure 1: \textit{C equivalent of the assembly routine \texttt{iir\_cas4}.}
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 1. This is the C equivalent of the hand-optimized assembly routine `iir_cas4()` taken from an older version of TI’s C62x DSPLIB.

   (a) Draw a signal flow graph of a part of the the routine’s code shown on the previous page; consider lines 4 to 7 only. Which filter is it?\(^2\)
   
   (b) Draw the block diagram of the filter represented by the code inside the routine’s `for`-loop.
   
   (c) Explain the meaning of a state in the given architecture.
   
   (d) What is the state space of the given digital filter and what’s the space’s dimension if the filter’s order is one?

2. Answer the following questions:

   (a) How many multiplications does the routine perform per input/output-sample? How many samples does the routine process during an iteration (an instant of time)?
   
   (b) Which numerical operation is equivalent to a 16-bit right-shift?
   
   (c) Why is it necessary to shift the states\(^3\) in this example?
   
   (d) Which numerical format (number of bits for the integer portion and for the fractional portion) is expected for the filter coefficients and for the input/output signal?

   \[
   \begin{array}{cccccccc}
   \text{N-1} & \text{N-2} & \text{N-3} & \text{N-4} & \text{N-5} & \ldots & 1 & 0 \\
   \text{S} & 2^\alpha & 2^\beta & 2^\gamma & 2^\delta & \ldots & 2^\psi & 2^\omega \\
   \text{SB} & \text{MSB} & \ldots & \text{LSB} \\
   \end{array}
   \]

   (e) Sometimes, non-linear effects and interferences may change the state of one or more bits accidently. The system may become unstable. What is the numerical range of the filter coefficients to ensure a stable filter behavior by considering the numerical format of the coefficients given in this experiment? Hints: (1) Write down the transfer function of the biquad filter and examine the denominator; (2) Assume complex conjugated poles and the numerical range of the filter’s coefficients calculated in (d).

   (f) Draw the area in the z-plane where poles can be placed. Does this routine represent an implementation of a general IIR filter?

   (g) How many bits do you have to consider at most for the right-shift to place poles anywhere inside the unit circle?

---

\(^2\)Note that a filter-architect may choose between different implementations, e.g., FIR/IIR, DF1/DF2, etc.

\(^3\)How many bits does the result of a summation and a multiplication of two 16-bit numbers exhibit? Which numerical formats exhibit both binary numbers?
Experiment 2
Limit Cycle due to ...?

Equipment: PC + DSK, headphones
Software: CCS, download [courses/dsplab/filt2/limcycl.zip] and unzip it on your workstation

1. Plug the output of the PC soundcard (or portable media player) to the DSK input and the headphones to the DSK output.

2. In CCS, load the provided project file limcyc1.pjt.

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

<table>
<thead>
<tr>
<th>b(0)</th>
<th>b(1)</th>
<th>b(2)</th>
<th>a(1)</th>
<th>a(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{15}$</td>
<td>0</td>
<td>0</td>
<td>$-Q_{15} \times \frac{3}{4}$</td>
<td>$Q_{15} \times \frac{3}{4}$</td>
</tr>
</tbody>
</table>

Draw an exact block-diagram of this system.

4. Build the program, load it to the DSP, and run it. Provide an input signal to the DSK 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.c again. Set the initial conditions to

<table>
<thead>
<tr>
<th>y(-2)</th>
<th>y(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-Q_{15} \times \frac{4}{5}$</td>
<td>$Q_{15} \times \frac{4}{5}$</td>
</tr>
</tbody>
</table>

6. Rebuild the program and load it to the DSP. 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 acoustically observe at the DSK’s output? Use an oscilloscope to specify the audible artifact.

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, remove main.c from the project and add main_rec.c instead. The difference of the new program is that the endless loop within main() has been replaced by a finite loop (as a consequence, the program will stop right after a few milliseconds) and the output of the filter will be recorded in the buffer short rec_buf[REC_BUF_SIZE]. Set the filter coefficient $b(0)=0$ in iir2.c to force zero input. Rebuild, reload, and run the program. It is not necessary to provide an input signal to the DSK now, because all feed-forward coefficients are zero.

9. Open a watch window to see the samples stored in the buffer rec_buf: select [View]→[Watch Window] enter the variable name rec_buf within the Watch 1 tab. Answer whether these samples can be the zero-input response of a stable linear system or not (explain). Analyze the source code in iir2.c and find out whether there is a nonlinearity that causes the observed behavior.
10. Edit the file `iir2.c` again. Activate the two lines that saturate the accumulator (just remove the comment characters).

11. Build and run the modified program and view the buffer again. What has changed?

References

