1 Introduction

This lab bulds upon labs 2 and 5 and, to some extent, lab 4, tying together individual design entities into a cohesive multirate DSP system.

The system samples a unipolar analog signal at 40 kHz using the MCP3202 ADC, interpolates the signal, stepping up its frequency, filters this higher frequency signal to remove the possibility of spectral aliasing and then decimates the signal to a lower frequency of 30 kHz before outputting it via the MCP4822 DAC chip.

2 Design Considerations

The system is arranged as a hierarchical structure, consisting of individually instantiated components connected together via port mapping and interconnecting signals where needed.

The system is pipelined, in the sense that multiple input signals propagate through different stages in the system at any given time. This pipelining is necessary for two reasons: the multirate nature of the system (as well as the imprecise clock generation, as discussed below) means that the input and output conversion triggers will drift relative to each other, necessitating this type of stage isolation. Additionally, transferring data over SPI is limited to the slowest SPI clock-rate, in this case 1 MHz. For a highest sampling rate of 40 kHz, or 25µs per sample, a theoretical maximum of 25 bits can be transferred. This leaves very little margin to serially read and write 12 bits (ie. without pipelining).

2.1 Clock generation

Each clocked component in the design requires a trigger signal in order to set its operating frequency and to control when it is to execute in the pipeline. Furthermore, an SPI clock must be generated between individual triggerings of the ADC/DAC driver components, and must be conditionally enabled in order to reduce the risk of errors (ie. only when there is data to transmit). This necessitates the use of multiple clocks, all derived from the system clock. Moreover, a trigger signal should only be one system clock cycle in length (T_{sys}), else multiple triggerings will occur, whereas the SPI clock outputted to the converter chips must have a 50% duty cycle.

In order to convert between frequencies in the discrete domain, frequencies must be changed by integer multiples. But, a conversion from 40 kHz to 30 kHz requires a ratio, namely 3/4. The solution is therefore to consider the numerator and denominator as individual interpolation and decimation operations, respectively. Interpolation must of course be performed before decimation to preserve the original signal information. The consequence of this two-step process is that an intermediate sample frequency is needed (120 kHz) and that this intermediate signal must be filtered to remove spectral artifacts resulting from the interpolation.

The actual clock frequencies that can be derived from the system clock will not be exactly accurate, however. To generate a frequency of 40 kHz, 100 MHz/40 kHz = 2500 system clock cycles are needed, which can be precisely achieved. 120 kHz on the other hand requires 100 MHz/120 kHz = 833,333 cycles, which is not realizable. As a compromise, 833 cycles are used, giving a frequency of approx. 120,048 kHz. This frequency can then be divided by four, giving 30,012 kHz, using 4 x 833 = 3332 cycles per period. The SPI clock operates at 1 MHz and is thus precisely realizable.

The clock generation module `clk_gen` is implemented in a fully generic manner and is used in all instances. The module takes generic parameters for target frequency (constrained to realizable values) and clock polarity (ie. whether a rising or falling edge occurs at $T_c/2$). The module, by default, outputs a main 50% duty clock of the target frequency and trigger pulses on both the falling and rising edges of this main clock. Usually, however, only one or two of these signals is required. The unwanted signals may be disabled in the instantiation as part of the port mapping, using the keywork `OPEN` as the destination. This disables the signal driver in the module instance.

2.2 ADC/DAC drivers

These components interface with the physical ADC/DAC chips and must therefore take into account external constraints.

The guiding principle behind the design of these components was to place data on the SPI data-line (MOSI) on the falling edge of the SPI clock (SCK), but read the input data-line (MISO) on the rising edge, in order to guarantee timing-margins given in the chip datasheets. In order to take action on either the rising or falling edge, corresponding trigger pulses were used, generated in the clock generation module.

2.3 Interpolation & Decimation

The interpolator was implemented as a circular FSM, clocked at the intermediate frequency (120 kHz), with a pass-through state in which it simply propagates the input value, and then two addition states where a zero is fed out. This essentially produces a "zero-stuffing" interpolator.

The decimator was simply implemented as a register, propagating the input value at the output sampling frequency (30 kHz).

2.4 LP FIR-filter

The serial FIR from lab 2 was reused for this design though the number of taps was changed in order to implement a satisfactory filter response. Variously, filter lengths of 10, 20 and 100 were tested, with a low number sufficing for the purposes of this lab where the passband characteristic was not of paramount importance.

The actual tap values were generated using an FIR-filter design tool at `http://t-filter.engineerjs.com/`, in decimal format. The tool allows the specification of passband and stop-band regions, as well as the desired number of taps. A custom MATLAB® script was then used to convert these taps to signed binary vectors. From there, minor text processing was used to insert the values into an appropriate VHDL array.

2.5 Datapath

The datapath is the top-level module in the hierarchy in which all individual component instances are interconnected. Some additional logic, not covered by existing modules, was required to achieve correct operation. Namely, the sampled unipolar signal must be converted to a signed representation before the FIR-filter receives it and then converted back afterwards. Fortunately, this could be achieved by simply inverting the signal MSB. It is important to do this before the interpolator, however, else the 'stuffed' zeros will be interpreted as the greatest negative value!

Additionally, the datapath is nominally 12 bits wide (to accomodate the format used in the ADC/DAC), but the FIR-filter outputs a wordlength of double this width (in order to maintain precision). Thus the 12 MSB's were propagated while the rest were discarded.

3 Simulation

Prior to the hardware implemenation, it was important to test for functional correctness. This was challenging, however, as the design interfaces with external components over SPI, making testbench design a non-trivial effort.

This was handled by designing (non-synthesizable) behavioural descriptions of both the physical ADC and DAC chips, that could take test-data and feed it over the SPI link, driven by the synthesizable part of the design.

Additionally, an unsigned, full-range sine signal was generated and provided to the aformentioned ADC component in order to simulate the analog input signal to the system. This module was

adapted from a dynamically generated design located at `https://www.doulos.com/knowhow/vhdl_designers_guide/models/sine_wave_generator/`. The module could also be varied in frequency in order to do some rudimentary tests of the filter response.

These simulation verification efforts saved considerable debugging and resulted in an immediately working hardware design at implementation-time.

4 Implementation & Verification

In the hardware implementation, slide-switches were connected to the configuration bits for the ADC/DAC drivers, mainly in order to select channel and output gain.

The post-implemention design did not strictly meet the timing requirements in the filter, giving a negative worst-case timing slack. One switch was therefore reserved as a filter bypass toggle, allowing the sampled signal to propagate unfiltered to the output. This turned out to be unneccessary, however, as no glitches were observed. On the other hand, this allowed us to observe the effects of aliasing when employing this type of frequency conversion.

To resolve the timing issues in the filter, an improvement might be to implement the filter MAC using distributed arithmetics. In this implementation, all multiplications are eliminated by precomputing them and storing them in an LUT. This should shorten the critical path significantly, with the added benefit of lowering the latency of operation from the number of taps to the wordlength. The downside is that the LUT size scales exponentially with the number of taps. For example, 20 tap constants give an LUT of 2^{20}=1048576 entries!

5 Additional Questions

5.1 *What problems might arise for a much larger sample rate change, such as a factor of ten? One Hundred?*

If increasing the frequency by a large amount, anti-aliasing filters (at the higher frequency) will have quite high requirements, as the cutoff frequency will be determined by the lower sample frequency. Also, in general, the higher the frequency, the less accurately it can be derived from the system clock. This is due to the relationship

$$f_{\text{derived}} = f_{\text{sys}}/\text{cycles per period} \tag{1}$$

which is asymptotic. Upon examination, the relationship is approximately linear (and therefore accurate) until 200 cycles (or 500 kHz in this case), and then becomes exponential as the number of cycles tends to zero.

5.2 *A common multirate system involves the conversion of audio signals from the CD standard sample rate of 44.1kHz to the Digital Audio Tape (DAT) sample rate of 48kHz, used by much studio equipment. If you were to use your solution for such a rate conversion, the internal sample rate would be quite high (how high?). Can you think of a way to ameliorate this situation?*

An interpolation/decimation ratio of 160/147 is required to convert 44.1 kHz to 48 kHz. The intermediate frequency is thus 44.1 kHz x 160 = 7.056 MHz. To avoid aliasing, this intermediate signal must be filtered with a cutoff frequency of 22.05, or $f_s/320$. This is an unrealistic design requirement.

A solution is to split the conversion into multiple stages, each with it's own interpolation, filtering and decimation. Stages with a net increase in frequency should precede those with a net decrease, to preserve signal information. In this case, the conversion ratio can be split into three parts

$$\frac{160}{147} = \frac{4}{3} \cdot \frac{8}{7} \cdot \frac{5}{7} \tag{2}$$

There will therefore be three intermediate frequencies, all with a lower value than before, ameliorating the demands on each individual filter.