Austrochip 2009
Tagungsband

7. Oktober 2009, Graz

Inhaltsverzeichnis

1. Vortragssession: Analogdesign und RFID

Multilevel Signaling Optical Receiver for High-Speed Transmission over Large-Core Step-Index Plastic Optical Fibre 1
M. Atef[1], W. Gaberl[1], Member, IEEE, R. Swoboda[2], H. Zimmermann[1], Senior Member, IEEE
[1] Vienna University of Technology, Institute of Electrodynamics, Microwave and Circuit Engineering
[2] A3PICs Electronics Development GmbH

A 65nm CMOS RF Power Detector with Integrated Offset Storage 5
Wolfgang Aichholzer, Johannes Sturm
Carinthia University of Applied Sciences

An 11-bit Successive Approximation Analog to Digital Converter Based on a Combined Capacitor-Resistor Network 9
Milos Davidovic, Gerald Zach, Horst Zimmermann
Vienna University of Technology, Institute of Electrodynamics, Microwave and Circuit Engineering

Design und Entwicklung eines Mixed Signal Prototyping Systems für RFID Applikationen mit Datenraten größer 848 kbit/s 15
Markus Auer[1], Edmund Ehrlich[1], Wolfgang Pribyl[1], Albert Missoni[2], Walter Kargl[2]
[1] Technische Universität Graz, Institut für Elektronik
[2] Infineon Technologies AG

2. Vortragssession: Digitaldesign

An IP-XACT Library extended with Verification Information for Functionality-based Component Selection 21
Christoph Ruggenthaler[1], Christoph Trummer[1], Christian Steger[1], Reinhold Weiß[1], Andreas Schuhai[2], Markus Pistauer[2], Damian Dalton[3],
[1]Institute for Technical Informatics, Graz University of Technology
[2]CISC Semiconductor Design+Consulting GmbH
[3]School of Computer Science and Informatics, University College Dublin

Investigating Power-Reduction for a Reconfigurable Sensor Interface 27
Johann Glaser, Jan Haase, Markus Damm, Christoph Grimm
Vienna University of Technology. Institute of Computer Technology

An Efficient FPGA Implementation of an Arbitrary Sampling Rate Converter for VoIP 33
Peter Brunmayr[1], Hans-Dieter Wohlmuth[2], Jan Haase[1]
[1]Institute of Computer Technology, Vienna University of Technology
[2]Frequentis AG

Weltweit kleinste, voll integrierte Lösung zur Uhrensynchronisation nach IEEE 1588 auf Schicht 2 39
R. Höller[1], H. Muhr[2], N. Kerö[2], A. Gröblinger[1], Ch. Kutschera[1], Ch. Veigl[1], Ch. Weiß[1], P. Rössler[1]
[1] Fachhochschule Technikum Wien
[2] Oregano Systems GmbH

3. Vortragssession: Modellierung/Design/EMV

A Behavioral Modeling Approach for Jitter Analysis in Charge-Pump PLLs 45
Stefan Erb, Wolfgang Pribyl
Institute of Electronics, Graz University of Technology

Comparative study of linear and non-linear integrated control schemes applied to a Buck converter for mobile applications 51
R. Priewasser[1], M. Agostinelli[1], S. Marsili[2], D. Straeussnigg[2], M. Huemer[1]
[1]Klagenfurt University
[2]Infineon Technologies Austria AG

Evaluating the Impact of using Fixed-Point Arithmetic on the Precision and Implementation of the Fit-to-Sine Algorithm on an FPGA 57
Andreas Weiss
EADS Defence and Security

Immunity Scan – Neuartige Untersuchungsmethode zum Lokalisieren von störempfindlichen Schaltungsblöcken in ICs 63
Dieter Maier, Bernhard Weiss, Rainer Minixhofer
austriamicrosystems AG

Beiträge für die Postersession

Compact Hardware Implementations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl, and Skein 69
Stefan Tillich, Martin Feldhofer, Wolfgang Issovits, Thomas Kern, Hermann Kureck, Michael Mühlberghuber, Georg Neubauer, Andreas Reiter, Armin Köfler, and Mathias Mayrhofer
Graz University of Technology, Institute for Applied Information Processing and Communications

Low-Power High-Speed Decimation Filter in 65 nm CMOS 75
Oleksandr Melnychenko, Sergii Zaiets, Manfred Ley
Fachhochschule Kärnten, School of Systems Engineering

Design Of A Fully Reconfigurable Bandpass Filter For Use In Cochlear Implants 81
Paul Faragó[1], Peter Söser[2], Sorin Hintea[1]
[1] Technical University of Cluj-Napoca
[2] Graz University of Technology

Stromquelle mit temperaturkompensiertem Referenzwiderstand 87
Stephan Dobretsberger
austriamicrosystems AG

A Flexible On-Chip Test Structure with Single Pin Serial Interface 91
Matthias Steiner
austriamicrosystems AG

Coupling Asynchronous Signals into Asynchronous Logic 97
Markus Ferringer
Institute of Computer Engineering, Embedded Computing Systems Group, Vienna University of Technology

Auswirkungen kapazitiver Kopplungen in Delta-Sigma-Modulatoren mit Switched-Capacitor-Feedback 103
Christoph B. Wurzinger
Technische Universität Graz, Institut für Elektronik

Hardware Acceleration of RNA Folding 109
Manuela Midl, Christian Netzberger
FH-Joanneum Kapfenberg

Using SRAMs as Physical Unclonable Functions 117
Christoph Böhm, Maximilian Hofer
Institute of Electronics, Graz University of Technology

A SystemC Design Pattern for the Cosimulation of Transaction-Level and Refined Cycle-Callable Models 123
Rainer Findenig[1], Wolfgang Ecker[2]
[1] FH Hagenberg
[2] Infineon Technologies AG

A Low-Voltage Charge Sampling Mixer for Direct Conversion at 1GHz in 65nm CMOS Technology 129
Kurt Schweiger, Horst Zimmermann
Institute of Electrodynamics, Microwave and Circuit Engineering, Vienna University of Technology

An Efficient FPGA Implementation of an Arbitrary Sampling Rate Converter for VoIP

Peter Brunmayr[1], Hans-Dieter Wohlmuth[2], Jan Haase[1]
[1]Institute of Computer Technology, Vienna University of Technology
[2]Frequentis AG, A-1100 Vienna, Austria
{brunmayr,haase}@ict.tuwien.ac.at,
hans-dieter.wohlmuth@frequentis.com

Abstract

In this paper, the implementation of a sample rate converter for arbitrary sampling rates is presented. The focus is especially on VoIP systems which are used for safety critical applications. In this systems, the sampling rates are nominally equal, but different clock sources cause slight differences. By using SystemC and high level synthesis it was possible to design a resampler with only one multiplier. The efficient design is small enough to fit several times on today's low-cost FPGAs. Simulation results show that audio signals are glitch-free converted between clock domains. The signal-to-noise ratio of the system is around 80 dB and thus in the area of the quantization noise of a 13 bit system.

1 Introduction

In voice over IP (VoIP) applications, the analog audio signal is converted to a digital signal and transmitted over an IP-based network. The receiver converts the received audio samples back to an analog signal. The sampling rate of the analog-to-digital converter (ADC) and of the digital-to-analog converter (DAC) first has to be negotiated between the sender and the receiver. Nevertheless, since the sender and the receiver are different systems at different locations, the desired sampling rate is generated from different clock sources. This leads to slightly different sampling rates.

The transmission of the samples over an IP-based network introduces a network jitter. To remove this jitter a buffer is necessary at the receiver. If the sampling rate of the receiver is a little bit slower than the sender's rate a buffer overflow will occur. If the sender's clock is slower, then a buffer underrun will occur at the receiver. In the first case, it is necessary to flush the whole buffer, which means that a part of the received audio signal is discarded. If an underrun occurs, silence is inserted. In both cases a disturbance will occur, which is not acceptable for safety critical communication systems used for air traffic management or in public transport systems.

PC-based VoIP systems reduce this problem by enlarging the buffer. This reduces the occurrence of the buffer overflows and underruns, but at the same time it enlarges the latency of the whole system. This is also not acceptable for safety critical systems. Thus, the buffer overflow and underrun problem is solved by recovering the sender's clock at the receiver site. Then, the jitter buffer can be read out with the sender's clock. In this way, a buffer overflow or underrun is avoided and the buffer can be kept very small. Anyway, it is necessary to convert the samples from the sender's clock domain to the receiver's domain. If no conversion is performed either at least one sample is taken twice or at least one sample is lost. With a typical frequency stability of 100 ppm [1] and a sampling rate of e.g. 8 kHz, every 1.25 seconds a short time broadband noise is introduced in the output signal. Of course, it is possible to use high accuracy oscillators with a better frequency stability, but this only reduces the problem. It does not solve it.

The operation of converting the signal between the clock domains is basically a resampling process. Since the sampling rates are almost equal and since the conversion ratio may change over time or from connection to connection, common resamplers can not be used. It is necessary to use special resamplers for arbitrary sampling rates, which are able to handle changes of the conversion ratio during runtime.

In this paper an efficient FPGA implementation of a resampling system is presented. The sampling rate converter has been optimized for a specific VoIP system. The boundary conditions are very typical for embedded VoIP systems and the resampler has been implemented using SystemC [2] and high level synthesis. Therefore, the design can easily be adapted for other systems. Also an integration in an ASIC should be possible with minor changes. One important design criterion was the overall delay of the filter. To fulfill the requirements for safety critical applications, the delay has to be as short as possible to ensure real time communication. For large systems with many channels, several resamplers have to be instantiated. Thus, a minimum usage of hardware resources was mandatory. Last but not least, the quality of the resampled audio is important. The additional noise introduced by the resampler has to be inaudible for humans.

The rest of this paper is structured as follows: In Section 2 different resampling methods are presented. The core component of the implemented method is an adaptive filter. The design of this filter is shown in Section 3. In Section 4 the implementation itself is explained in detail. The used design flow is shown in Section 5 Finally, synthesis and simulation results are presented in Section 6 and the work is concluded in Section 7.

2 Conversion between arbitrary sampling frequencies

Resampling a signal to a new sampling frequency denotes the process of calculating new sample values in between the original samples. If the relation between the sampling rates is a rational factor, resampling is usually performed by interpolating the signal by an integer factor, filtering the signal to avoid aliasing and finally decimating the signal by another integer factor [7]. To use this method, the ratio has to be known in advance.

In this work, the resampler has to convert between two almost equal sampling frequencies and the ratio itself may be different for every VoIP call. Therefore, it is necessary that the system adapts itself to every new ratio. There are several different methods presented in [3] to convert a signal between arbitrary sampling frequencies. A simple method has first been presented in [6]. Lagadec *et al.* proposed to interpolate the signal to a very high frequency. The output signal is then generated by taking the closest sample to the correct sampling instant. The higher the sampling frequency F_s, to which the signal is interpolated, the smaller is the error in the output signal. Ramstad has shown in [3] that the error in the output signal is smaller than the quantization error if the inequation

$$F_s \geq \pi \cdot 2^{b+1} F_M \tag{1}$$

is fulfilled, F_M denoting the highest frequency of the signal and b denoting the bit width. Many VoIP systems use pulse code modulation (PCM) like G.711 [9] with a sampling rate of 8 kHz. These systems encode 13 bit samples to 8 bit with a logarithmic characteristic. It is assumed in this paper that the signal is band limited with a maximum frequency of 4 kHz. According to Eqn. 1 this band limitation leads to an interpolation frequency of about 200 MHz for a 13 bit system.

If linear interpolation between the neighboring samples is included, as presented in [4], the inequation changes to

$$F_s \geq \pi \cdot 2^{(b+1)/2} F_M \tag{2}$$

according to [3]. This reduces the needed interpolation frequency to about 1.6 MHz, but the computation effort for the interpolation filter would still be tremendous.

A different approach is presented by Smith in [5]. Suppose we have a digital signal $x(nT_s)$ with a sampling frequency $F_s = 1/T_s$, n ranges over the integers, which is assumed to be bandlimited to the half of the sampling frequency. Due to Shannon's sampling theorem it is possible to reconstruct the original signal using

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT_s) h_s(t - nT_s), \tag{3}$$

where

$$h_s(t) = sinc(F_s t) = \frac{sin(\pi F_s t)}{\pi F_s t}. \tag{4}$$

Eq. 3 basically denotes a convolution of the digital signal with a continuous Sinc function. The Fourier transform of the Sinc function is a rectangle. Thus, the convolution of $x(t)$ and $h_s(t)$ corresponds with a filtering process with an ideal low pass filter $H_s(f)$ in the frequency domain. Thereby, the image spectra of the periodic spectrum $X(f)$ are removed and the original spectrum of the continuous signal is reconstructed, as shown in Fig. 1.

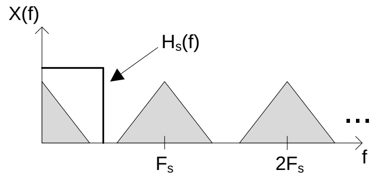

Figure 1. Ideal reconstruction of the continuous signal from a band limited digital signal.

If this signal has to be resampled to the sampling frequency $F_s' = 1/T_s'$, then Eq. 3 only has to be evaluated at the sampling instants of the new sampling frequency

$$x(mT_s') = \sum_{n=-\infty}^{\infty} x(nT_s) h_s(mT_s' - nT_s). \tag{5}$$

3 Filter Design

Using Eq. 5 the signal can be resampled ideally. Obviously, it is not realizable, since it implies an infinite sum. It is necessary to design a digital filter with a finite impulse response (FIR), which is used as $h_s(t)$. Several methods exist to design digital FIR filters [8]. For our purpose it is necessary to highly oversample the filter to get an almost continuous impulse response. According to [5], the window method using the Kaiser window is a very simple and robust method ideal for high sampling frequencies [15].

To find the ideal sampling frequency for the filter, different design constraints have to be kept in mind. On the one hand, the frequency has to be high enough, so that the error due to the discrete impulse response is smaller or equal to the quantization error. On the other hand, if a higher sampling frequency is used, a larger memory is needed to save all the coefficients. The accuracy of the filter coefficients can be increased by a linear interpolation during runtime. This significantly reduces the needed sampling frequency. In principle, the sampling frequency of the impulse response can be interpreted as the sampling frequency, to which the signal is first interpolated before it is sampled with the new sampling rate. The difference is, that output values are only calculated if they are needed. Therefore, Eq. 2 can be used to estimate the needed frequency. To simplify the implementation, the interpolation frequency also should be an integer multiple of the sampling frequency of the audio data.

Our VoIP target system operates with a 50 MHz system clock. The data is PCM encoded using the a-law codec. Thus, the samples have an accuracy comparable to 13 bit linear data. The sampling frequency is 8 kHz. Using Eq. 2 leads to a minimum sampling frequency of 1.6 MHz for the filter. This calculation is overly pessimistic. For simplification the sampling frequency is set to 1 MHz, which is an integer multiple of the sampling frequency of the audio data.

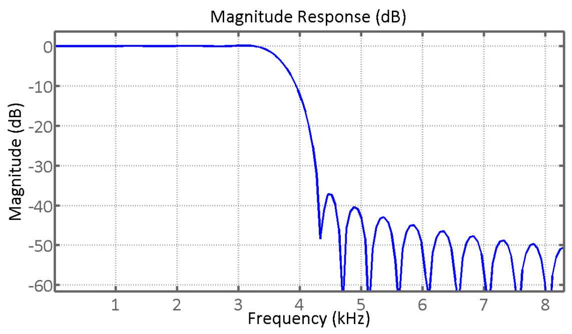

Figure 2. Magnitude response of the designed filter function.

Another design decision is the length of the impulse response. With a longer impulse response, a steeper filter with a larger stop band attenuation can be designed. However, a longer impulse response increases the overall latency of the filter. Since this work focuses on safety critical communication systems, the latency should be as small as possible.

The implemented impulse response is a filter with the order 2002. With a sampling frequency of 1 MHz the filter has a latency of approximately 1 millisecond. The filter has been designed with the window method using the Kaiser window. Due to the ITU-T recommendation G.712 [10] the transition band can start at 3.4 kHz. We assume that aliasing filters already attenuate the frequency band between 3.4 and 4 kHz. Thus, we can enlarge the transition band to 4.6 kHz. The cut-off frequency is 3800 kHz and the beta value of the Kaiser method is 3. This value influences the style of the Kaiser window. A larger beta value enlarges the transition band of the filter, but it also reduces the ripple in the stop band. The magnitude response of the filter is shown in Fig. 2. Pass band ripples are smaller than 1 dB. At 4.6 kHz the attenuation is around 40 dB.

4 Implementation

The basic structure of the implementation of the resampling engine can be seen in Fig. 3. The entity has four inputs, a data input, a data output for the resampled data and two clock inputs. The data at the input is decoded. Thus, the data ports have a bit width of 13 bits. As data format a two's complement signed fixed point data format with one bit before the comma and 12 fractional bits is used. This kind of fixed point data format is very common in signal processing applications. Since numbers in the range of $[-1;1[$ can be expressed, multiplications only lead to an overflow if both factors are -1. The input clock is the recovered clock from the sender, with which the samples are read out of the jitter buffer and the output clock is the clock of the DAC at the receiver. There are two other inputs, which are not shown in this high level representation, the system clock and the reset. The whole design is synchronous to the system clock and all other clock signals are treated as data signals.

With every rising edge of the input clock one new sample is written to the dual port RAM. The input clock is also routed to the time measurement unit, which measures the time since the last rising edge of the input clock. This information together with the output clock can be used by the adaptive filter to calculate the current phase difference of the clocks. The phase difference is then used to load the correct coefficients out of the coefficient ROM and to calculate a new output value out of several old input values from the dual port RAM.

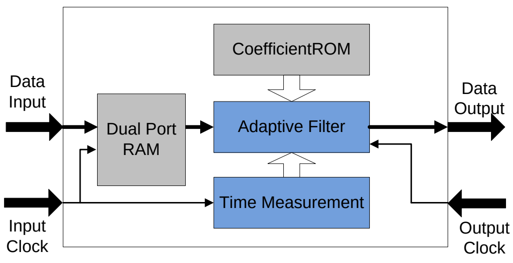

Figure 3. Basic structure of the resampling engine.

The coefficient ROM holds the impulse response designed in Section 3. It is basically a Sinc function which has been modified by a Kaiser window. The resulting function is still symmetric. Thus, it is enough to save only one half of the impulse response. This results in only 1002 coefficients, which have to be saved. With a bit width of 12 bits this leads to 1.5 kbytes of needed memory.

The dual port RAM, which holds the input values is organized as a ring buffer. After 32 input values, the oldest input value is overwritten by a new value. Every time a new value is written, a pointer register is updated. It points to the memory cell which is written next.

This pointer register is also needed by the adaptive filter. Each time a rising edge occurs at the output clock, the adaptive filter saves the current value of the pointer register and the current value of the time register t of the time measurement unit. This time value denotes the phase difference between the input and the output clock.

In Fig. 4 the input values $x[n]$ are shown on a time axis. The position n of the middle value $x[n]$ can be derived from the pointer register. For each output value 16 input values are needed. In other words, the filter has 16 tabs. With an order of 2000 and a sampling rate of 1 MHz a length of the impulse response of about 2 milliseconds can be derived. The sampling frequency of the data is 8 kHz. Hence, the distance between the input samples is $L =$

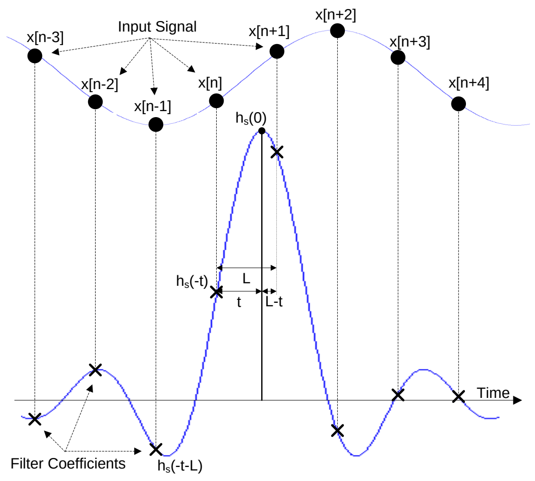

Figure 4. Input values and corresponding coefficients.

125 μs, which explains the 16 tabs. The Sinc function is shifted according to the phase difference t. In this way, the correct coefficient for the central input value $x[n]$ can be found at $h_s(-t) = h_s(t)$. The neighboring coefficients can be found by adding the data's period $h_s(t + L)$.

$$y = \sum_{i=0}^{8} x[n-i] \left[h_s[t_1 + iL] + t_2 \cdot \overline{h}_s(t_1 + iL)\right] \quad (6)$$

$$y = y + \sum_{i=0}^{8} x[n+1+i] \left[h_s[t_1 + iL] + t_2 \cdot \overline{h}_s(t_1 + iL)\right] . \quad (7)$$

For the calculation of the next output value y two sums have to be implemented, Eq. 6 and Eq. 7. First, one half of the Sinc function is applied to the older input values, all values left of the central input value $x[n]$, see Fig. 4. Then, the newer input values are multiplied with its coefficients and the results are accumulated.

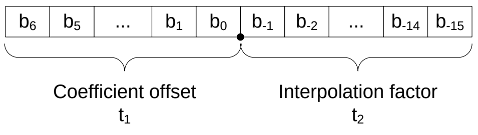

Figure 5. Binary format of time register.

To load the correct coefficient out of the coefficient table $h_s[n]$ in the ROM, the time register t is needed. Its format is shown in Fig. 5. It consists of an integer part t_1 and of a fractional part t_2. The unit of the time value is microseconds. Therefore, the integer part denotes the offset for loading the coefficient and the fractional part is then used for the linear interpolation. To interpolate between two neighboring coefficients, the used coefficient $h_s[t_1 + iL]$ and the next coefficient $h_s[t_1 + iL + 1]$ are loaded and the difference

$$\overline{h}_s(t_1 + iL) = h_s[t_1 + iL + 1] - h_s[t_1 + iL] \quad (8)$$

is calculated. This difference is then multiplied with the fractional part of the time register t_2. For the second sum, the time register has to be modified by

$$t = L - t \quad (9)$$

so that $h_s(t)$ equates the coefficient of input value $x[n+1]$.

As already mentioned, the time register consists of a fractional and an integer part and it represents the time since the last rising edge of the input clock (in microseconds). This data format, shown in Fig. 5, simplifies the separation into coefficient offset and fractional factor. The time is measured in the time measurement unit, see Fig. 3, with a simple counter. For the system clock of 50 MHz 0.02 has to be added for each clock cycle. This is not representable in a binary data format with a finite number of bits. A timing error is introduced by the finite binary representation of 0.02. The bit width has to be large enough, so that the noise introduced by the timing error is below the quantization noise.

The accuracy of the time value influences the calculation of the linear interpolation of the coefficients. The linear interpolation is performed by multiplying the difference of two neighboring coefficients with the fractional part of the time. The difference of two coefficients is always smaller than 2^{-6} and the data format uses 12 fractional bits. Consequently, an accuracy of the fractional part t_2 of the time until the bit 2^{-6} is enough to not affect the quantized result of the linear interpolation. Thus, the time error should be smaller than 2^{-6} μs $=$ 0.0156 μs. With 15 fractional bits the number 0.0200042724609375 can be expressed. The maximum time error occurs if the phase difference is almost one sampling period. One sampling period consists of 125 μs / 0.02 μs $=$ 65,250 system clock cycles. Hence, the worst case error is around 0.0267 μs.

5 Design flow

The first step in the design flow is a high level implementation of the algorithm. Therefore, the resampler has been implemented as a so-called SystemC functional model [11]. The functional model consists of a pure software implementation of the algorithm encapsulated in a SystemC module, which already has the same interfaces, called ports, as the later implementation. In this way it was possible to evaluate the algorithm already with the sampling rates and clock frequencies of the target system. To analyze the signal quality in Matlab [12], a library has been implemented to read and write audio wave files from the SystemC simulation.

For the translation from a high level SystemC description to register transfer level (RTL) Verilog code, the high level synthesis (HLS) tool Cynthesizer from Forte Design Systems [13] has been used. To use HLS it is necessary to refine the code step-by-step. After each step, e.g. conversion from floating point to fixed point, the implementation has been simulated and the quality of the output signal has been measured again. This simulation framework allows the immediate evaluation of design decisions. In other words, it is possible to try different implementations within the design space. One possible implementation uses pre-computed differences of neighboring coefficients. These differences can then be saved in a table to reduce the calculation effort during runtime. Tests have shown that the reduction of the hardware is too low compared to the additional table which is needed.

Listing 1. High level implementation of Eq. 6

```
for (int i=0; i<(NrOfTabs/2); i++)
{
    sc_fixed<13,1> Interpolation =
    FracTime*(cNum[IntTime]-cNum[IntTime+1]);
    Output += Buffer[Position]*
    (cNum[IntTime] + Interpolation);

    IntTime += sc_uint<11>(125);
    Position++;
}
```

The refined SystemC code can then be synthesized to RTL Verilog code. In this step there are additional possibilities to test different design variants. The abstraction level of the code is still very high. The core algorithm, e.g. the first sum, see Listing 1, is completely untimed and besides the fixed point data types the code is pure C/C++. Such a high level code only describes the behavior, not the structure of the hardware. Therefore, there are many different hardware structures which implement the same behavior. The HLS tool tries to find the optimal design under consideration of user defined design constraints. Using these design constraints it is possible to generate implementations with e.g. different numbers of multipliers or with different latencies.

The generated Verilog code has then been simulated again with the original test framework. After that the code has then been implemented using the logic synthesis tool XST from Xilinx.

The presented design flow has several advantages. As already mentioned, design space exploration is simplified. It is easier to test different implementations on a high abstraction level than on register transfer level and also the synthesis tool supports design space exploration by just changing design constraints. Additionally, the development time has been reduced significantly. The resampling engine uses a few hardware components sequentially. To design such an optimized data path and the associated controller with Verilog or VHDL needs a lot more development effort and experienced hardware designers.

Nevertheless, there are also disadvantages. The HLS tools are still not fully developed. The hardware estimations were very often not accurate enough to use them to evaluate different design decisions. Thus, HLS and logic synthesis were necessary to get accurate estimations of the number of used hardware resources. This, indeed, increased the time to evaluate different design solutions.

Another problem is that better HLS tools mainly focus on ASIC design. The poor support for FPGAs is evident if special components like block RAMs or embedded hard macros are to be used. They can not be instantiated with standardized SystemC code. Proprietary code structures are necessary to synthesis designs with such components.

6 Results

The resample engine has been successfully tested with the VoIP target system. The design has been synthesized for a Spartan 3A 3s400aft256 device with a clock frequency of 50 MHz. The required resources are shown in Tab. 1

Table 1. Hardware resources of resampler engine.

Resources	#	Usage in [%]
Slices	232	6
Slice Flip Flops	261	3
4 input LUTs	435	6
BRAMs	2	10
Multipliers	1	5

The resampling engine only needs six percent of the logic of the FPGA. The design computes the filter function sequentially and therefore only one multiplier is required. All coefficients fit into one block RAM, which has a size of 18 kbits. The other block RAM is used as dual port RAM for the input values.

The whole system is small enough to be instantiated several times on one FPGA, which would be necessary to resample different audio channels. If the channels have to be converted between the same clock domains, additional optimizations can be done. The phase difference between the two clock domains has to be calculated only once and can be used for all channels. Since the sampling period is very large, it is possible to compute many channels sequentially. In this case one block RAM is still enough to save the input values of all channels. For channels operating in the same direction it is even possible to reuse the calculated coefficient.

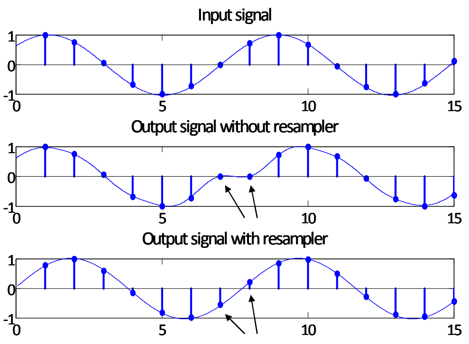

Figure 6. Input signal vs. output signal without resampler (showing errors) and with resampler.

The RTL code generated by the HLS tool has been simulated with the original SystemC test bench. Fig. 6 shows a section of two output signals. The first signal has been generated by converting a sine with 1020 Hz from one clock domain to another without any resampler. The second part of the Figure shows the output signal, when the resampling engine is used. In the first output signal one sample is taken twice. If the resampler is used, this glitch is removed completely.

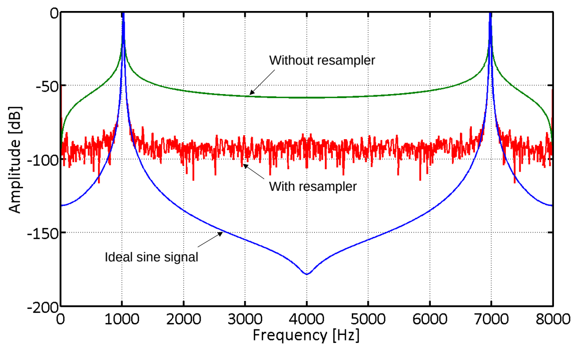

Figure 7. Spectrum of output signal without and with resampler vs. ideal sine signal.

Fig. 7 shows the fast Fourier transform (FFT) of a 128 ms sequence of three audio signals. The lower curve with the least distortions is the spectrum of an ideal sine signal with double precision generated in Matlab. It also includes portions of other frequencies. The reason is that a finite sequence has to be cut out of the signal to calculate the FFT and the length of this sequence does not correspond with a period of the signal. In the second audio signal one sample is taken twice. The spectrum of this signal, see the upper curve in Fig. 7, has a very high noise floor at around -55 dB. With the resampling engine, the glitch is removed completely, but the permanent noise floor is increased to around 80 dB. The spectrum of the resampled signal is the middle curve in Fig. 7. Compared to the glitch, the signal-to-noise ratio (SNR) is improved by about 25 dB.

7 Conclusion

This work shows the design and implementation of a digital sampling rate converter for arbitrary sampling rates. By using design space exploration it was possible to design an optimized data path with the associated controller. The design exploits the large periods of audio applications by performing many calculations sequentially. The implementation can be used for multi channel applications, since it fits several times on one low-cost FPGA. The use of SystemC and high level synthesis has simplified the design space exploration and has reduced the overall development time. Nevertheless, there are still disadvantages of this design technique, which have to be solved to increase its acceptance for digital hardware design. The results show that the short time glitch is removed completely. The permanent noise of 80 dB, which is introduced, is around the quantization noise of a 13 bit system.

Possible next steps could be to optimize the design for several channels. As already mentioned, many components can be reused and the hardware effort can be reduced dramatically. Another goal could be to improve the reusability of the design, so that it can more easily be adapted for other boundary conditions.

References

[1] Abracon Corporation, "Datasheet: Crystal Clock Oscillators ACOL and ACHL," http://www.abracon.com/Oscillators/acol-achl.pdf

[2] Open SystemC Initiative, "SystemC^{TM}," http://www.systemc.org

[3] T.A. Ramstad, "Digital Methods for Conversion Between Arbitrary Sampling Frequencies," *IEEE Trans. on Acoustics, Speech and Signal Processing*, Vol. 32, pp. 577 - 591, Jun 1984

[4] R. Lagadec and H.O. Kunz, "A universal, digital sampling frequency converter for digital audio," *IEEE Int. Conference on Acoustics, Speech and Signal Processing*, Vol. 6, pp. 595 - 598, Apr 1981

[5] J.O. Smith, "Digital Audio Resampling Home Page," Jan 2002 http://www-ccrma.stanford.edu/~jos/resample/

[6] R. Lagadec, D. Pelloni, and D. Weiss, "A 2-Channel, 16-bit digital sampling frequency converter for professional digital audio," *IEEE Int. Conference on Acoustics, Speech and Signal Processing*, Vol. 7, pp. 93 - 96, May 1982

[7] R.E. Crochiere and L.R. Rabiner, "Multirate Digital Signal Processing," Prentice Hall, 1983

[8] A.V. Oppenheim, R.W. Schafer, and J.R. Buck, "Discrete-Time Signal Processing," Prentice Hall, 1999

[9] International Telecommunication Union, "G.711: Pulse code modulation (PCM) of voice frequencies," http://www.itu.int/rec/T-REC-G.711/e;

[10] International Telecommunication Union, "G.712: Transmission performance characteristic of pulse code modulation channels,"; http://www.itu.int/rec/T-REC-G.712-200111-I/en;

[11] P. Brunmayr, J. Haase, and F. Schupfer, "Late Hardware/-Software Partitioning by using SystemC Functional Models," *Proc. of the 3rd Asia Int. Converence on Modelling & Simulation (AMS 2009)*, pp. 194 - 199

[12] The Mathworks, Inc, "Matlab," http://www.mathworks.com

[13] Forte Design Systems, "Cynthesizer," http://www.forteds.com

[14] Xilinx, Inc, http://www.xilinx.com

[15] J.F. Kaiser and R.W. Schafer, "On the Use of the I_0-Sinh Window for Spectrum Analysis" *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 28, 1980, pp. 105 - 107