A Fractional Sample Rate Converter With Parallelized Multiphase Output: Algorithm and FPGA Implementation

Shahriar Shahabuddin[1,3*], Petri Manninen[2] and Markku Juntti[3]

[1*]Mobile Networks, Nokia, Kaapelitie 4, Oulu, 90620, Finland.
[2]Digital Enhancement Suomi Oy, Kiviharjunlenkki 1 B, Oulu, 90220, Finland.
[3]Centre for Wireless Communications, University of Oulu, Erkki Koiso-Kanttilan katu 3, Oulu, 90570, Finland.

*Corresponding author(s). E-mail(s): shahriar.shahabuddin@nokia.com;
Contributing authors: petri.m.manninen@gmail.com;
markku.juntti@oulu.fi;

Abstract

Sample rate conversion is an essential scheme used in almost every radio design. Supporting sampling rates higher than the clock rates require parallel processing. In this paper, we propose an algorithm for a sample rate converter (SRC) with multiple parallel output phases so that the conversion ratio can be a fixed rational number. Due to the structure of the proposed algorithm, it is suitable for embedded platforms which are restricted by their clock frequency but require very high sample rates. A dual phase output variant of the proposed algorithm is simulated with a 400 MHz input signal to perform a $\mathbf{15/8}$ conversion. The test and verification of the SRC algorithm is presented with the aid of a design example. A VLSI architecture of the dual phase output SRC is implemented on a Virtex-7 field-programmable gate array (FPGA) and results are presented.

Keywords: SRC, polyphase, filter, parallel, multiphase

1 Introduction

Sample rate conversion is a process to change the effective sample rate of a discrete-time signal [1]. A sample rate converter (SRC) is essential in real-time processing when two hardware units operating at different sample rates exchange digital data. The application of SRC can be found nearly in every digital radio now-a-days. SRC can be used to decrease or increase the sample rates. The operations are known as decimation or interpolation, respectively. If a conversion with a fraction M/D of a rational number is required, interpolation by an integer factor M followed by a decimation by an integer factor of D is usually performed. This process is illustrated in Fig. 1. A combination of up-sampling and down-sampling operations, denoted by $\uparrow M$ and $\downarrow D$ respectively, is used to change the sampling rate of a discrete sequence $x(k)$ to a new set of discrete sequence $y(k)$.

The drawback of down-sampling is the aliasing effect and an anti-aliasing filter, denoted by LPF_D in Fig. 1(a), is typically applied before the down-sampler. The up-sampling operation produces unwanted spectra and an anti-imaging filter, denoted by LPF_M in Fig. 1(a), is typically used after the up-sampler. A single low-pass filter, LPF_{MD} can be used to combine the anti-aliasing filter and anti-imaging filters [2]. Any tool to design a low-pass filter can be applied to this task [1]. A straightforward implementation of the fractional SRC is not efficient. Firstly, the sampling rate is increased by M by inserting $M-1$ zeroes between every input sample. The filter has to calculate outputs at a high sampling rate, computing all zero inputs in vain, and for every output sample, $D-1$ samples are discarded. To overcome these inefficiencies, a polyphase structure is typically used for fractional SRCs [3–6]. The polyphase SRC can be implemented in a straightforward manner as long as the hardware does not cause restrictions for the structure.

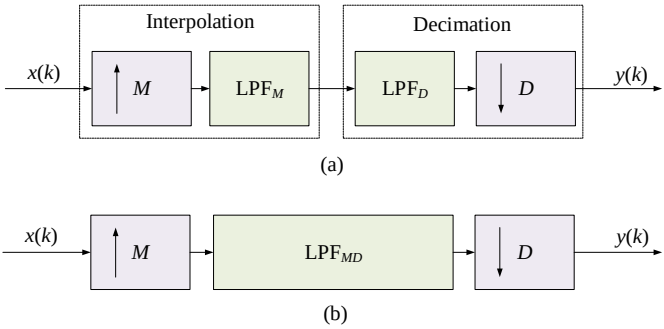

Fig. 1: Block diagram of a fractional sample rate converter. (a) The cascade of a traditional interpolator and decimator. (b) A single filter replacing the anti-imaging and anti-aliasing filters.

A major restriction comes from the maximum operating clock frequency of a hardware platform. For example, a clock in a field-programmable gate array (FPGA)

system is responsible for driving the FPGA design and determines how fast the flip-flops toggle. A faster operating clock translates into faster data processing in any FPGA system [7]. Typically, a large digital design with high resource consumption can achieve roughly 500 to 600 MHz maximum operating clock frequency in modern FPGAs. In the context of SRC, this means an SRC interpolator can provide 500 to 600 Msps at the output for such a design. However, due to demand for ever-increasing data rates and signal bandwidths of next-generation wireless systems, the output sample rate required at the output of an SRC filter might be significantly higher than the clock rate. For example, millimeter wave communication utilizes very high carrier bandwidth which requires very high sample rate processing [8]. This type of filters, where sample frequency is higher than the clock frequency, is known as super sample rate filters [9].

One way to circumvent this limitation is parallelizing the input data samples and apply conventional SRC in each parallel path [10, 11]. We take a different approach and design an SRC which receives a serial input stream and provide an output sample rate higher than the clock rate. If the SRC output sample rate is an integer multiple M of the input sample rate, then samples can be easily interpolated and divided into M parallel paths. The M paths would represent the M phases of the original signal. We invite interested readers to go through [12] to read more about parallel structures of polyphase filters. However, for a non-integer sample rate conversion ratio a fractional SRC is needed, and for parallel outputs, a novel algorithm and realization is required. Note that, most state-of-the-art fractional SRC architectures do not support super sample rate [13–17]. As an application-specific integrated circuit (ASIC) can achieve significantly higher operating clock frequency than an FPGA design, SRC ASICs are out of the scope of this work.

In this paper, we propose a fractional SRC algorithm that works for a fixed set of sampling frequencies and can provide parallelized multiphase outputs. The output of the conventional SRC is divided into N phases and the resulting implementation is based on N parallel polyphase structures with N times shorter subfilters. Thus, the overall complexity is the same as the original filter, but with an increased number of parallel processing branches. A fixed point simulation of the SRC algorithm is presented for a 400 MHz passband signal to fulfill -60 dB stopband and 0.1 dB ripple. Finally, the algorithm is implemented in a Xilinx Virtex-7 FPGA platform to estimate resource utilization.

The paper is organized as follows: A generic mathematical model for SRC algorithm based on polyphase structure is presented in Section 2. From single phase output SRC, the mathematical model for multiphase output is derived and an algorithm suitable for dual phase output is presented in Section 3. The multiphase SRC is presented in algorithm format in Section 4. The detail of the fixed-point simulation and FPGA implementation are presented in Sections 5 and 6 respectively. The conclusions are drawn in Section 7.

2 SRC based on Polyphase FIR Filters

In order to describe parallelized SRC algorithm, we need to first go through mathematical basics of a normal polyphase SRC. Let us assume that we have to obtain an output sample rate of f_{os} from an input sampling rate of f_{is} such that

$$f_{os} = (M/D)f_{is}, \tag{1}$$

where M and D are positive, relatively prime integers. We also assume a straightforward finite impulse response (FIR) filter optimized for this sample rate conversion requires R taps $h_M(n)$, $n \in \{0, 1, \ldots, R-1\}$. After up-sampling by M and filtering, and before down-sampling, the intermediate signal y_M is

$$y_M(k) = \sum_{i=0}^{R-1} x_M(k-i)h(i), \tag{2}$$

where x_M is the up-sampled input signal defined as

$$x_M(kM+j) = \begin{cases} x(k), & j = 0 \\ 0, & j \in \{1, 2, \ldots, M-1\}. \end{cases} \tag{3}$$

The filtering process in (2) calculates the full convolution between the up-sampled signal and filter taps. However, since the up-sampled signal includes zeros, the convolution can be shortened to include only the non-zero samples, which are the same as the original input samples $x(n)$. The length L of the needed convolution is

$$L = \lceil \frac{R}{M} \rceil. \tag{4}$$

Thereby, the filtering process can be reformulated as

$$y_M(kM+j) = \sum_{i=0}^{L-1} x(k-i)h(iM+j). \tag{5}$$

In the down-sampling process, only every Dth intermediate sample is taken as an output: $y(k) = y_M(kD)$. Therefore, only every Dth intermediate sample needs to be calculated. For this, new indices j' and k' can be defined as a function of k so that $k'(k)M + j'(k) = kD$, i.e., $y_M(k'(k)M + j'(k)) = y_M(kD)$. Since j' is modular with modulus M, the solution comes from the Euclidian division, where j' is the remainder and k' the quotient of kD divided by M as

$$j'(k) = kD \bmod M; \tag{6}$$

$$k'(k) = \lfloor \frac{kD}{M} \rfloor. \tag{7}$$

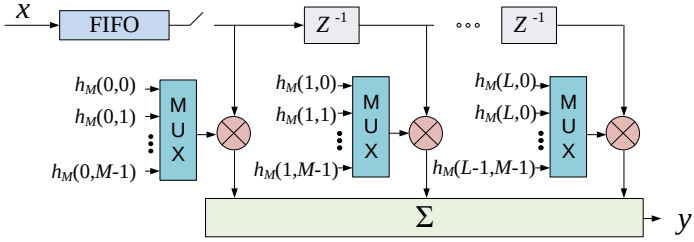

Fig. 2: Commutator model of SRC.

The output y of the entire SRC becomes

$$\begin{aligned} y(k) &= \sum_{i=0}^{L-1} x\big(k'(k)-i\big)h\big(iM+j'(k)\big) \\ &= \sum_{i=0}^{L-1} x\big(\lfloor \tfrac{kD}{M} \rfloor - i\big)h\big(iM+(kD \bmod M)\big). \end{aligned} \tag{8}$$

To further clarify the design, we present an SRC structure based on the commutator model in Fig. 2, where the co-efficient sets are stored and selected by a multiplexer to efficiently reuse the multipliers. The multiplexers are denoted by MUX, the multipliers are denoted by a cross in a circle, and a rectangle with a summation sign denotes the addition operations performed on the output of the multipliers in Fig. 2.

In a practical implementation, there can be a counter j' that is incremented by D for every output sample, and it wraps around with modulus M. For every wrap-around, delay line of x values is shifted by one; that is equivalent to k' being incremented by one. It should be noticed that in case $D > M$ there can be more than one "wrap-around" in one step. So, actually when the counter j' is incremented, the quotient of $(j'+D)/M$ defines the number of delay line shifts, and only after that the remainder is set as new counter value. In this sense, wrap-around is somewhat inadequate term and must be used with caution.

Addressing one large memory unit containing all filter taps $h(n)$ is not practical and usually not even feasible. It is better to divide the filter into M *phases*; the length of each phase specific filter is L, and the phase is selected by j'. The taps to the phase specific filter sets h_M are selected from h as

$$h_M(i,j') = h(iM+j'), \tag{9}$$

where i runs from 0 to $L-1$ for each $j' \in \{0,1,\ldots,M-1\}$. Filter $h(n)$ should be designed so that the length R is an integer multiple of M. Thereby all the filter sets h_M are fully utilized. Otherwise, $(LM-R)$ zeros need to be appended to $h(n)$.

3 Multiphase Output SRC based on Polyphase FIR

Following the derivations in Section 2, we formulate the algorithm for multiphase output in this section. Continuing from (6) - (8), we divide the output samples $y(n)$ into several output phases $y_p(n)$

$$y_p(k) = y(kN + p), \tag{10}$$

where N is the number of output phases, $p \in \{0, 1, \ldots, N-1\}$. Let $D_N = ND$ denote overall decimation factor. Before combining (8) with (10), let us define a new decimation factor D_N such that

$$D_N = ND \tag{11}$$

Using (8) in (10), the output $y_p(k)$ becomes

$$y_p(k) = \sum_{i=0}^{L-1} x\Big(\lfloor \tfrac{(kD_N+pD}{M} \rfloor - i\Big) h\Big(iM + \big((kD_N + pD) \bmod M\big)\Big). \tag{12}$$

Using similar notation as that in (8), (12) can be written as

$$y_p(k) = \sum_{i=0}^{L-1} x\big(k_p(k) - i\big) h\big(iM + j_p(k)\big), \tag{13}$$

where j_p and k_p are

$$j_p(k) = (kD_N + pD) \bmod M; \tag{14}$$

$$k_p(k) = \lfloor \frac{kD_N + pD}{M} \rfloor. \tag{15}$$

Similar to the conventional polyphase SRC, in an implementation there can be a counter j_p that is incremented by D_N for every output sample so that it wraps around with modulus M. However, there must be a separate counter for each output phase p, and the counters start with an offset depending on the phase: the offset is pD. Also the delay lines of $x(n)$ values should be separate for each phase and shifted according to the wrap-around(s) by their own counter. To further clarify the design, the block diagram of a SRC structure with two parallel outputs is presented in Fig. 3. Two polyphase filters working in parallel on the same data stream is producing two sets of outputs in Fig. 3.

Similar to single data rate SRC, the filter taps can be divided into sets h_M. In general, there is no difference in defining the sets compared to the earlier method. Set h_M can be defined as in (9), i.e., the contents are the same and in the same order. The filter phases are just being addressed by j_p instead of j' separately for each output phase, as $h_M(i, j_p)$.

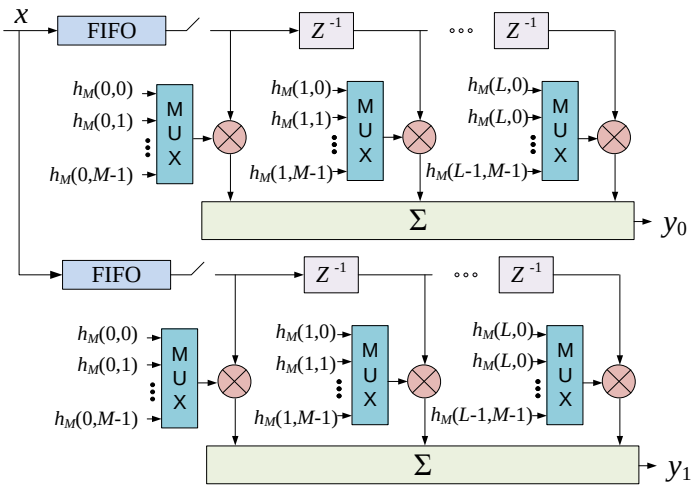

Fig. 3: Realization of a SRC with two output phases. Generic approach is assumed for the filter sets.

However, if M is an integer multiple of the number of output phases N, each phase will use only part of the filter sets; namely, each phase uses M/N sets. The selection of sets for each phase can be done calculating all values for $j_p(k)$ in (14) using $k \in \{0, 1, \ldots, M/N - 1\}$.

Hence, in general case the memory for filter sets need to be replicated for each phase. Only if M happens to be an integer multiple of N it is possible to reduce memory usage. Unfortunately, HW designer typically cannot select M value since it depends on the whole system. Therefore, for generic approach we must assume the memory size cannot be optimized. Hence, in every sense the multiphase output requires truly parallel SRC structures for each phase and there is not much that can be done to avoid it. For further reference, we rewrite (13) using the filter sets h_M as

$$y_p(k) = \sum_{i=0}^{L-1} x\big(k_p(k) - i\big) h_M\big(i, j_p(k)\big). \tag{16}$$

4 Multiphase Output SRC Algorithm for Implementation

A simplified algorithm for fractional SRC with two parallel outputs is presented in Algorithm 1. We refer this use case double data rate (DDR) SRC throughout the paper, meaning SRC generates two output phases. We create two identical $M \times L$ polyphase tap matrices $\mathbf{H}_a$ and $\mathbf{H}_b$ from the input FIR filter taps, $\mathbf{h}$. The same $S_1 \times 1$ input sample vector $\mathbf{x}$ is fed to both paths of DDR SRC to generate two separate $S_2 \times 1$

output sample vectors $\mathbf{y}_1$ and $\mathbf{y}_2$, where the value of S_2 can be determined as

$$S_2 = \lceil (S_1 + L - 1)\frac{M}{2D} \rceil \tag{17}$$

Contrary to single data rate SRC, we need to select two different rows of $\mathbf{H}_a$ and $\mathbf{H}_b$ respectively at the same time and select the appropriate input data stream for each path. For example, if we select the first row of $\mathbf{H}_a$, we need to select the second row of $\mathbf{H}_b$. Therefore, we need to initialize the index for phases with $p_1 = 1$ and $p_2 = 2$ respectively and increment them by a value of 2. If M is an odd number, we need to store all the taps for both paths. For example, if there are 5 rows, the first, third and fifth row will be selected by p_1 and the second and forth row will be selected by p_2. However, in the next round, p_2 will select the odd rows and p_1 will select the even rows. Thus, both path will require all the taps. If M is an even number, then we can divide the taps in odd rows and even rows between two paths and save half of the memory.

The next step is to select an appropriate M consecutive inputs samples from $\mathbf{x}$. A modulo counter with size of M, which is incremented by D, is used to select the starting position in each cycle. After each in increment we check how many times the counter wraps around and this is the amount of delay line shifts. The residual is kept for the next round. Two accumulators A_1 and A_2 are used for two paths which are initialized as $A_1 = 0$ and $A_2 = D$ respectively.

Algorithm 1 Proposed DDR SRC Algorithm

Input: $\mathbf{x} \in \mathbb{R}^{S_1 \times 1}, \mathbf{H}_a, \mathbf{H}_b \in \mathbb{R}^{M \times L}, M, D \in \mathbb{Z}$
Output: $\mathbf{y}_1 \in \mathbb{R}^{S_2 \times 1}, \mathbf{y}_2 \in \mathbb{R}^{S_2 \times 1}$
Initialize phase and accumulators:
$p_1 = 1, p_2 = 2$
$k_{p1} = 0, k_{p2} = D$
Index of the first input samples in FIR:
$i_1 = 1, i_2 = 1$
Main Loop:
for $k = 1 : S_2$
  $\mathbf{y}_1(k) = \mathbf{H}_a(p1, :) * \mathbf{x}(i_1 : i_1 + L - 1)$
  $\mathbf{y}_2(k) = \mathbf{H}_b(p2, :) * \mathbf{x}(i_2 : i_2 + L - 1)$
  $p_1 = ((p_1 + 1) \bmod M) + 1$
  $p_2 = ((p_2 + 1) \bmod M) + 1$
  $j_{p1} = \lfloor (k_{p1} + 2D)/M \rfloor$
  $j_{p2} = \lfloor (k_{p2} + 2D)/M \rfloor$
  $k_{p1} = k_{p1} + 2D - Mj_{p1}$
  $k_{p2} = k_{p2} + 2D - Mj_{p2}$
  $i_1 = i_1 + j_{p1}$
  $i_2 = i_2 + j_{p2}$
end

We present the complete DDR SRC algorithm in 1 which can be used for implementation. The algorithm takes $\mathbf{x} \in \mathbb{R}^{L\times 1}$, $\mathbf{h} \in \mathbb{R}^{R\times 1}$, $M \in \mathbb{Z}$ and $D \in \mathbb{Z}$ as inputs. The algorithm provides $\mathbf{y}_1 \in \mathbb{R}^{S\times 1}$ and $\mathbf{y}_2 \in \mathbb{R}^{S\times 1}$ as outputs. The polyphase filter tap matrix is formed by line 1 - 9 in the algorithm. The size of the output vectors are determined in line 10. The DDR structure is presented from line 11 - 25.

The phase, accumulators and index of the first input samples are initialized in line 1 - 3. In the main loop, line 5 and 6 are doing the calculations for multiplying filter tap matrix and the discrete input stream and the indexing is derived in the subsequent lines. Line 7 and 8 are used to update the phase indexes used to select appropriate rows of filter tap matrix. Line 9 and 10 calculates the shift of the delay line which can also be calculated as $j_{p1} = (k_{p1} + 2D) \bmod M$ and $j_{p2} = (k_{p2} + 2D) \bmod M$. The calculation required to update the accumulator is provided in line 11 and 12. Line 13 and 14 is used for updating the indexing here while in real implementation this would be a shift of delay line. We would like to note that it is possible to support more output phases. However, it will increase complexity significantly.

5 Performance of Multiphase Output SRC

The optimal design of a filter depends on the required bandwidth and alias attenuation. Initially, we calculate the taps for a straightforward FIR implementation. The virtual sample rate for the polyphase filter is Mf_{is} according to the definition in (1). If the bandwidth of the signal is W, then the one-sided passband edge is at $W/2$ and the stopband starts at $f_{is} - W/2$ or $f_{os} - W/2$, whichever is lower. In practice, it might be needed to include a small margin to the passband and/or stopband edges. These values can then be used to compute the filter taps h, for example by least-squares solution [18] or by equal ripple Parks-McClellan optimization [19, 20].

We assume a signal bandwidth of $W = 400$ MHz and a sample rate of $f_{is} = 491.52$ Msps that needs to be converted to $f_{os} = 921.6$ Msps. Using parallelized DDR SRC shown earlier the output is divided to two phases, each having sampling rate of 460.8 Msps. So, no clock above 491.52 MHz is needed in the system. The conversion ratio of $921.6/491.52$ can be expressed with relatively prime integers as $15/8$, i.e. the interpolation factor $M = 15$ and the decimation factor $D = 8$. We also assume constraints of minimum 60 dB stopband attenuation and passband ripple of highest ± 0.1 dB. The least-squares FIR filter designer in Matlab [21, 22] generates a total of 269 taps for $h(n)$ that satisfies the attenuation and ripple constraints. The number 269 is the nearest odd number to 270 that is an integer multiple of 15. A performance comparison of the FIR with different number of taps is presented in Table 1.

Table 1: Performance comparison of the FIR

taps	stopband attenuation	transition width	passband ripple
269	-62.28 dB	44.4 MHz	0.06663 dB
224	-56.3 dB	45.3 MHz	0.18383 dB
149	-44.95 dB	49.8 MHz	0.72876 dB
90	-36.8 dB	59.7 MHz	1.622 dB

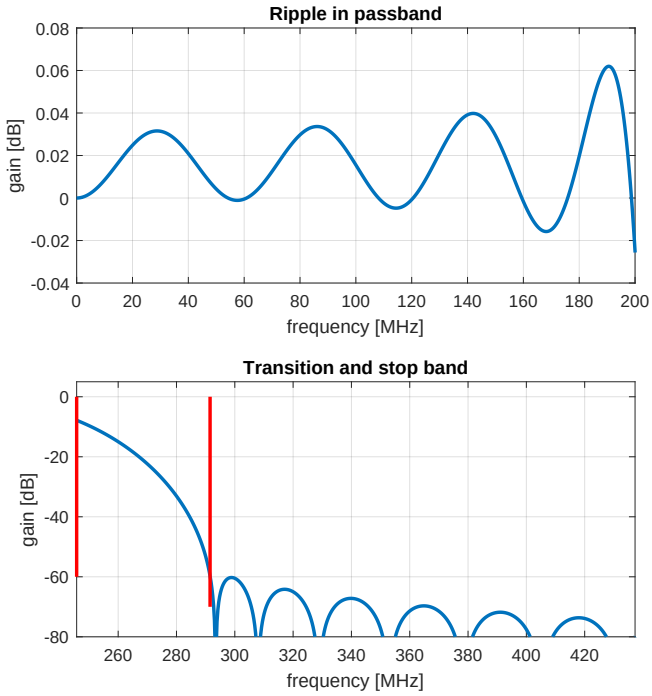

Fig. 4: Passband ripple, transition and stopband of the optimized filter h.

The passband ripple, transition band and the stopband are shown in Fig. 4. It is evident that the least-squares optimized filter is able to contain the passband ripple within ± 0.1 dB. The stopband starts well before the vertical red line that marks the required stopband edge, and the minimum attenuation is slightly over 60 dB. The figures are obtained by applying a fast Fourier transform on h extended with zeros to improve the spectral accuracy.

The output spectrum of the SRC is shown in Fig. 5. The input signal has rectangular shaped 400 MHz bandwidth and 491.52 Msps sampling rate. The output signal has the same shape and bandwidth with 921.6 Msps sampling rate. The maximum power spectral density (PSD) of noise is 60 dB below the PSD of the signal, that is consistent with the stopband attenuation. Note that, the complex input samples are quantized to a total of 32-bits, where the real and imaginary parts are 16-bits each. The filter co-efficients are quantized to 14-bits. We notice that there is no significant degradation in the output spectrum of the SRC due to the quantization.

It should be noted that the samples of two output paths are combined into one signal to generate the total spectrum. In the implementation, every second output sample belongs to another path. In Fig. 6, the input and (combined) output samples are shown together with ideally[1] interpolated input signal (dashed line). The output

[1]Ideal interpolation can be done by FFT for a cyclic signal.

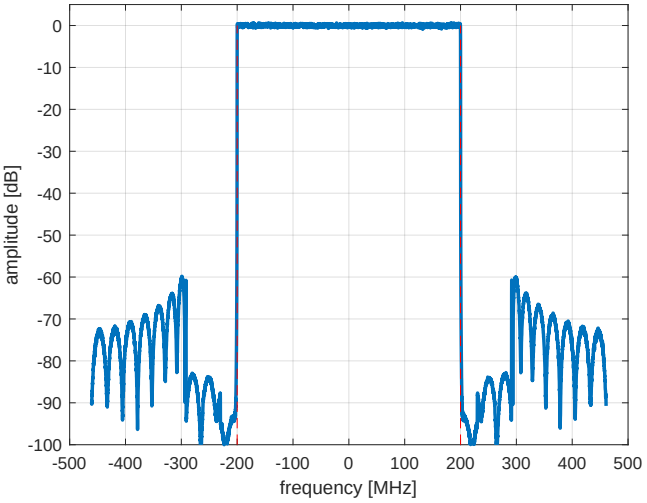

Fig. 5: DDR SRC output with 400 MHz input signal.

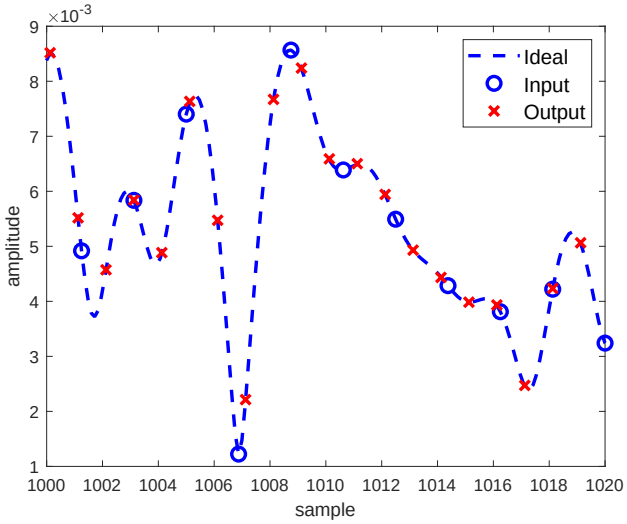

Fig. 6: SRC input and output samples compared with the ideally interpolated input.

samples follow the ideally interpolated signal, and every 15th output sample matches every 8th input sample that demonstrates the conversion ratio of $15/8$.

Table 2: Component-wise breakdown in Virtex-7

Components	Without BRAM	With BRAM
LUT Slices	2185	577
FF	756	756
DSP units	104	104
BRAM	0	4
I/O	60	60

Table 3: Comparison of Resource Consumption of different SRC implementations on FPGA

year and reference	**algorithm**	**device**	**LUT Slices**	**FF**	**DSP**	**power (W)**
2014 [10]	parallel FIR	Xilinx Virtex-7	25826	44547	660	7.44
2014 [13]	CIC and Farrow	Xilinx Virtex-6	50646	28234	-	-
2015 [11]	parallel polyphase	Xilinx Virtex-7	9792	12240	1872	-
2017 [14]	CIC and polyphase	Xilinx Kintex-7	7269	13552	83	1.446
2019 [15]	CIC and FIR	Xilinx Kintex-7	-	1882	-	1.022
2020 [16]	CIC and FIR	Xilinx Kintex-7	1828	2253	148	0.325
2021 [17]	CIC and polyphase	Xilinx Kintex-7	1268	1850	40	0.270
Proposed	Polyphase FIR	Xilinx Virtex-7	2185	756	104	0.510

6 FPGA Implementation

In this section, we present the VLSI architecture and the FPGA implementation results of DDR SRC for the use case of Section 5. An intuitive block diagram of the DDR SRC VLSI architecture can be found in Fig. 7. Based on the simulation results in the preceding section, the inputs of the architecture are 14-bit coefficients, h and 32-bit samples, x. Due to the polyphase structure of our filter, we would need to access only $L = 18$ coefficients at a time instant. The same coefficients are stored in two separate random access memories (RAM) in such a way that one read access makes $L = 18$ coefficients corresponding to a phase available to the multiplier array. Therefore, the width of a word for a RAM is $14 \times L$. As the size of the word is quite large, an intermediate register bank is used to receive the L coefficients sequentially and concatenate them to store as a word in a RAM.

On the other hand, the complex input samples x are sequentially stored in a shift register based first-in first-out (FIFO) memory. The length of the FIFO buffer is taken sufficiently large so that the memory does not overflow. To our advantage, we need to read consecutive values from the shift-register based FIFO arrays. Two sets of consecutive L samples are required to be read from the FIFO to feed as an input of the multiplier array. However, the starting point of these consecutive sets depends on lines 9 and 10 of Algorithm 1. This corresponds to the shift in the delay line which happens at lines 13 and 14. The calculations for j_p values require division according

Table 4: Comparison of different SRC implementations on FPGA

year and reference	algorithm	advantage	disadvantage
2014 [10]	parallel FIR	supports Gigabit data rates and fractional SRC	parallel input streams required, very high resource usage
2014 [13]	CIC and Farrow	supports fractional SRC and arbitrary conversion ratio	very high resource usage, only supports 3G, no super sample rate[a] support
2015 [11]	parallel polyphase	applied for high speed optical communication, supports fractional SRC	parallel input streams required
2017 [14]	CIC and polyphase	highly configurable design, supports fractional SRC and arbitrary conversion ratio	high power consumption, no support for super sample rates[a]
2019 [15]	CIC and FIR	number of slice registers less than [14] for same decimation factor	Impact on number of LUTs and DSPs are unclear, no super sample rate[a] support
2020 [16]	CIC and FIR	rapid design presented with the aid of high level synthesis	no support for super sample rates[a] and arbitrary conversion ratio
2021 [17]	CIC and polyphase	Reduce resource and power consumption	no support for super sample rates[a] and arbitrary conversion ratio
Proposed	Polyphase FIR	Works with serial input streams, supports super sample rates[a] and 5G	no support for arbitrary conversion ratio

[a]Super sample rate filters support sample frequency greater than clock frequency [9]

to Algorithm 1. The complex division operation can be avoided by simplifying the j_p calculations of lines 9 and 10 for our particular use case as

$$j_{p1} = \begin{cases} 2, & \text{when } k_{p1} \geq 2M \\ 1, & \text{when } 2M > k_{p1} \geq M \\ 0 & \text{otherwise} \end{cases}$$

We can compute j_{p2} in a similar fashion with the aid of k_{p2}. Therefore, instead of using the modulo operations of line 9 and 10, we design the j_{p1} and j_{p2} with simple multiplexing logic. The calculation of j_{p1} and j_{p2} are shown as read pointer blocks in Fig. 7.

For reading the coefficients from the RAM, we have to use lines 7 and 8 of Algorithm 1. We use two counters for $p1$ and $p2$ which wraps around at M to select the addresses from the RAM. These counters are expressed as address selection logic in Fig. 7. The counters are initialized with different values as shown in line 1 of Algorithm 1. Hence, we need to have separate counters and separate single-port RAMs in the design. Note that, it is also possible to use only one dual-port RAM instead of separate RAMs with the same content. Two arrays of multipliers are used for the design where each array contains a total of L real-complex multipliers. A summation is required in a filtering operation and therefore, the output of the L multipliers are

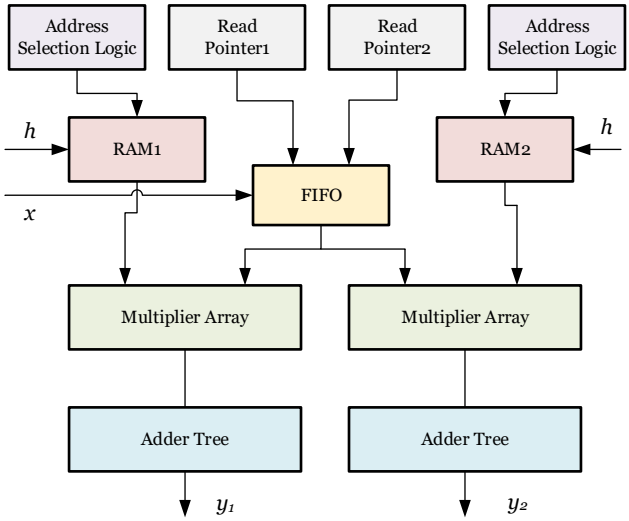

Fig. 7: Top level VLSI architecture for DDR SRC.

summed with a complex adder tree. The multiplier array and adder tree are registered at each level to aid the timing closure.

The VLSI architecture for the DDR SRC is written with VHSIC hardware description language (VHDL). As we do not use any technology-specific Xilinx primitives, the code is highly portable to FPGA platforms. The VHDL code is synthesized and implemented on a Xilinx Virtex-7 XC7VX690T FPGA. We apply Vivado default settings as the synthesis and implementation strategy. The default mode is selected for the `-flatten_hierarchy` option in the Vivado design tool to keep the same top-level hierarchy after synthesis. We synthesize the DDR SRC with a target clock frequency of 495 MHz. The component wise breakdown of the DDR SRC implementation is presented in Table 2. The table shows the number of look-up tables (LUT), flip-flops (FF), DSP slices and I/O pin used for DDR SRC with BRAMs and without BRAMs. Due to the synthesis settings, the Vivado tool uses the LUT slices for memory in the left column of Table 2. The DSP slices represent the real-complex multipliers and the complex adder tree logic. The right column presents the result when the BRAM tiles of the FPGA fabric are utilized. Depending on the availability of LUT slices and BRAMs, one of the implementation choices can be selected. For example, if an FPGA implementation of the baseband transceiver has already utilized most of the LUTs, the DDR SRC of column two can be selected. Our work can be further extended to quad data rate (QDR) SRC with four parallel outputs to provide 1966.08 Msps.

In Table 3, we present hardware costs of state-of-the-art SRC FPGA implementations and compare with our implementation. We also compare their pros and cons in

Table 4. In [10], the authors presented a parallel sample rate for the high speed backhaul networks. The authors verified their design with an incoming parallel stream of 1.7 Giga-samples-per-second (Gsps) which provided an interpolated signal at 2.8 Gsps. However, the authors assumed that the incoming stream of 1.7 Gsps is already parallelized into several low rate data streams which might not be always available to the designers. The authors applied conventional SRC on each of the parallel streams. On the contrary, our proposed algorithm and design work on serial inputs and provide parallel output to achieve a super sample rate. The implementation in [10] utilizes about $10\times$ higher number of LUTs and $60\times$ higher number of FFs than our design. In [13], the authors presented an SRC filter for software radio receiver. As the target of the design was a flexible digital front-end in the intermediate frequency (IF) stage of a software defined radio, the authors designed a multiplexed Cascaded Integrator Comb (CIC) decimation filter. The authors also applied a reconfigurable Farrow filter for droop compensation. Due to the application of Farrow structure, the design can support arbitrary sample rate conversion. However, it comes with a price of very high resource usage due to the complexity of the Farrow filter. The design also does not support super sample rate like our proposed solution. In [11], the authors presented a parallel fractional SRC for high speed optical communications. The authors applied a conventional polyphase FIR filter for the SRC operation. However, the design also assumed parallel input streams similar to [10] and applied conventional SRCs in parallel. The implementation of [11] requires about $16\times$ higher number of FFs and $6\times$ higher numbers of DSPs than our design.

A digital down converter (DDC) is proposed in [14] which uses a combination of half-band filter, CIC filter, polyphase filter, and a Farrow-based variable fractional delay (VFD) filter to downconvert the input signal in stages. Due to the application of a Farrow filter, the design can support an arbitrary sample rate. However, due to several complex filters is used in the design, the design requires a higher number of resources and consumes $3\times$ more power than our design. In [15], the authors proposed another DDC based on a combination of COordinate Rotation Digital Computer (CORDIC) processor, CIC, and FIR filter. Due to the use of a CORDIC processor, the resource consumption is significantly less than [14] for similar functionalities. A couple of other recent DDC implementations can be found in [16] and [17], which are also comparable to our design in terms of resource consumption. However, none of [15–17] supports super sample rates, i.e. clock rate lower than sample rates, like our design.

Contrary to the existing implementations, we consider a realistic use case for fifth generation (5G) millimeter-wave systems with a high bandwidth carrier of 400 MHz. For example, the parallel SRC of [13] was presented for third generation (3G) technologies such as WiMAX and GSM900 standards. As the 5G advanced and sixth generation (6G) systems are moving towards higher frequencies and bandwidth, we believe our work will be very useful for the communication engineers and circuit designers working in this domain. SRC are typically used in the front-end of a base station (BS). However, the front-end of a BS typically also includes digital predistortion, crest factor reduction, and other important blocks. Therefore, it might be challenging to accommodate an SRC with many parallel paths in a BS front-end FPGA.

7 Conclusions

In this paper, we presented a generic approach for a fractional SRC with multiphase output. This solves the conversion problem when the sampling rate needs to be higher than the clock rate, and the conversion rate is not an integer number. Clock rate limitation is especially significant in FPGA design; therefore, an efficient realization from an FPGA point of view was considered. The performance of the DDR SRC was presented with the aid of a design example. Finally, a VLSI architecture and corresponding FPGA implementation were presented for DDR SRC.

References

[1] Lyons, R.G.: Understanding Digital Signal Processing. Prentice Hall, Englewood Cliffs, NJ, USA (2004)

[2] Milic, L.: Multirate Filtering for Digital Signal Processing: MATLAB Applications. IGI Global, Hershey, PA, USA (2009)

[3] Vaidyanathan, P.P.: Multirate Systems and Filter Banks. Prentice Hall, Englewood Cliffs, NJ, USA (1993)

[4] Zeineddine, A., Nafkha, A., Paquelet, S., Moy, C., Jezequel, P.Y.: Comprehensive survey of FIR-based sample rate conversion. Journal of Signal Processing Systems **93**(1), 113–125 (2021)

[5] Evangelista, G.: Design of digital systems for arbitrary sampling rate conversion. Signal processing **83**(2), 377–387 (2003)

[6] Göckler, H.G., Evangelista, G., Groth, A.: Minimal block processing approach to fractional sample rate conversion. Signal processing **81**(4), 673–691 (2001)

[7] Gandhare, S., Karthikeyan, B.: Survey on FPGA architecture and recent applications. In: 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), pp. 1–4 (2019). IEEE

[8] Heath, R.W., Gonzalez-Prelcic, N., Rangan, S., Roh, W., Sayeed, A.M.: An overview of signal processing techniques for millimeter wave MIMO systems. IEEE journal of selected topics in signal processing **10**(3), 436–453 (2016)

[9] Xilinx: FIR Compiler v7.2 LogiCORE IP Product Guide. Vivado Design Suit **PG149** (2021)

[10] Alonso, A., Sevillano, J.F., Vélez, I.: Parallel implementation of a sample rate conversion and pulse-shaping filter for high speed backhauling networks. In: IEEE Design of Circuits and Integrated Systems, pp. 1–6 (2014)

[11] Sousa, I., Boas, B.V., Freire, I., Klautau, A., Reis, J.D.: Parallel polyphase

filtering for pulse shaping on high-speed optical communication systems. In: IEEE International Microwave and Optoelectronics Conference, pp. 1–5 (2015)

[12] Parhi, K.K.: VLSI Digital Signal Processing Systems: Design and Implementation. John Wiley & Sons, Hoboken, NJ, USA (2007)

[13] Agarwal, A., Boppana, L., Kodali, R.K.: A fractional sample rate conversion filter for a software radio receiver on FPGA. In: IEEE Region 10 Conference (TENCON), pp. 1–6 (2014)

[14] Liu, X., Yan, X.-X., Wang, Z.-K., Deng, Q.-X.: Design and FPGA implementation of a reconfigurable digital down converter for wideband applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**(12), 3548–3552 (2017)

[15] Datta, D., Mitra, P., Dutta, H.S.: FPGA implementation of high performance digital down converter for software defined radio. Microsystem Technologies, 1–10 (2019)

[16] Sikka, P., Asati, A.R., Shekhar, C.: Power-and area-optimized high-level synthesis implementation of a digital down converter for software-defined radio applications. Circuits, Systems, and Signal Processing, 1–12 (2020)

[17] Datta, D., Dutta, H.S.: High efficient polyphase digital down converter on FPGA. Circuits, Systems, and Signal Processing, 1–12 (2021)

[18] Adams, J.W.: FIR digital filters with least-squares stopbands subject to peak-gain constraints. IEEE Transactions on circuits and systems **38**(4), 376–388 (1991)

[19] Parks, T., McClellan, J.: Chebyshev approximation for nonrecursive digital filters with linear phase. IEEE Transactions on Circuit Theory **19**(2), 189–194 (1972)

[20] McClellan, J.H., Parks, T.W.: A personal history of the Parks-McClellan algorithm. IEEE signal processing magazine **22**(2), 82–86 (2005)

[21] Losada, R.A.: Practical FIR filter design in MATLAB. The Math Works inc. Revision **1**, 5–26 (2004)

[22] Stearns, S.D., Hush, D.R.: Digital Signal Processing with Examples in MATLAB®. CRC Press, Boca Raton, FL, USA (2016)