Simulink/Matlab-to-VHDL Route for Full-Custom/FPGA Rapid Prototyping of DSP Algorithms

Artur KRUKOWSKI and Izzet KALE

University of Westminster, United Kingdom.

Abstract

This paper presents the way of speeding up the route from the theoretical design with Simulink/Matlab, via behavioral simulation in fixed-point arithmetic to the implementation on either FPGA or custom silicon. This has been achieved by porting the netlist of the Simulink system description into the VHDL. At the first instance, the Simulink-to-VHDL converter has been designed to use structural VHDL code to describe system interconnections, allowing simple behavioral descriptions for basic blocks. The resulting VHDL code delivers bit-true result when compared to the equivalent fixed-point Simulink model simulations.

1. Introduction

However, the success of VHDL for designing integrated circuits is indisputable. Unfortunately there is a lack of tools available linking VHDL tools with such high-level digital filter design/simulation tools like Matlab™ and Simulink™, which operate on the levels higher than the structure. At the moment the designer who designed and tested his design theoretically using high-level tools is required to spend the same or more time on designing the structure and the architecture for his theoretical design, simulate it, test it and fabricate it. This involves a dangerous break in the integrity of design flow, giving chances for inconsistencies to creep in. An automated high-integrity link between theoretical design and implementation is essential and can be achieved with VHDL via a conversion tool. A very attractive high-level design/simulation tool is provided by MathWorks™ and is called Simulink™. It is a very flexible design tool, which allows testing of a high-level structural description of the design and makes possible quick changes and corrections. The circuit description structure is very similar to the way the design could be implemented later. Therefore mapping tool allowing conversion of such a structure into VHDL code would save the designer's time, which otherwise has to be spent in rewriting the same structure in VHDL and probably making mistakes that will need debugging. This idea is the basis of the work described in this paper.

Primarily, the work has been concentrated on the analysis of the Simulink™ structure and its similarity with the VHDL description. The structural style of programming has been chosen for the first version of our conversion tool, as this would allow direct mapping of Simulink™ structures into ones described in VHDL. As Simulink™ is a high-level description tool and allows such operations as unconstrained arithmetic operations, the behavioural style will be included in the next version of the conversion tool. The ultimate tool to be developed will also allow incorporation of some form of simple optimisation into the mapping process.

2. Basics of VHDL

VHDL stands for Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (HDL). It is a language for describing digital electronic systems. It was born out of the United States Government's VHSIC program in 1980 and was adopted as a standard for describing the structure and function of Integrated Circuits (IC). Soon after it was developed and adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US (IEEE-1076-1987) and in other countries [1,2]. VHDL continues to evolve. Although new standards have been prepared (VHDL-93) most commercial VHDL tools use 1076-1987 version of VHDL, thus making it the most compatible when using different compilation tools. The 1076-1987 standard has also been used here.

VHDL enables the designer to:

- Describe the design in its structure, to specify how it is decomposed into sub-designs, and how these sub-designs are interconnected.
- Specify the function of designs using a familiar, C-like programming language form.
- Simulate the design before sending it off for fabrication, so that the designer has a chance to rapidly compare alternative approach and test for correctness without the delay and expense of multiple prototyping.

VHDL is a C-like, general purpose programming language with extensions to model both concurrent and

sequential flows of execution, and allowing delayed assignment of values. To a first approximation VHDL can be considered to be a combination of two languages: one describing the structure of the integrated circuit and its interconnections (structural description) and the other one describing its behaviour using algorithmic constructs (behavioural description).

VHDL allows three styles of programming:

1. Structural
2. Register Transfer Level
3. Behavioural

The first one, structural, is the most commonly used as it allows description of the structure of the IC very precisely by the user. This in very many cases gives the best performance over compiler optimised structures, especially for high speed, fixed-point applications like polyphase IIR structures [5-8]. Its behavioural style permits the designer to quickly test concepts, where the designer can specify the high-level function of the design without taking much care how it will be done structurally. This can be very attractive for quick design of low and medium-speed and low-volume applications, where the designer expertise is not available. A word of warning is appropriate here. Designs synthesised from behavioural descriptions will often end up using a lot more resources than actually necessary, even after optimisation.

2.1. Effective Implementation via Simulink-to-VHDL Conversion Tool.

So far the biggest problem which the designer faces very often is how to pass from the algorithmic design to its physical implementation. The first tool the designer uses when developing the new idea is a high-level design and simulation tool. One of the most commonly used high-level tools is Matlab™ with Simulink™. It allows the designer to put together a behavioural or structural simulation very easily and quickly checking the algorithm or making the necessary adjustments to it. Working directly with any low-level implementation tool from the start is simply not practical, as every small change in the algorithm may sometimes require substantial redesign of the implementation. Therefore an automatic link between the high-level algorithmic design, like Simulink™ model, to some implementation description, like a target netlist or VHDL, would lead to great effort and time savings in the design cycle.

Matlab™ has been used at the University of Westminster, Applied DSP Research Group, for a long time and has proven to be an invaluable tool for DSP applications. Therefore this software was chosen for the high-level design part of the whole system. In the first instance Simulink™ has been chosen to be the input to the conversion tool. The fact that Simulink™ makes it possible to design both behavioural and structural designs (where this latter one is the closest to the physical implementation) justifies its choice. The description of a typical Simulink™ block is similar to the netlist of the physical implementation. The VHDL description has been chosen for the output of the conversion tool, as it is the highest level technology-independent description of the design to be realised. There are also many tools available both Unix and PC based for compiling VHDL into a netlist, then ported into the custom silicon fabrication arena or FPGAs. Such tools include Peak VHDL/FPGA from Accolade Design Automation Inc. [3], Galileo and Renoir from Mentor [4].

The tasks of the converter can be described as follows:

1. Analyse the Simulink™ model and identify:
   - Common and different blocks
   - Connections (signals) and ports for multilevel models
   - Block parameters
2. Generate a VHDL equivalent:
   - Find entities available in standard component library
   - Create architectures for each block from bottom up
   - Create configuration files for every entity linking in standard libraries

It can be easily noticed is that there is a set of blocks in Simulink™, which have to be treated as the basic ones. There are compiled "s-functions", the contents of that are not available. Therefore, their behaviour has to be carefully analysed in order to create their equivalent VHDL descriptions, to be later included into the library of standard Simulink™ entities/architectures.

2.2. Basics of Simulink.

Simulink™, as is true for most of high-level simulation software, does not allow testing certain behaviour patterns that a real target design can exhibit, most of which are available for the VHDL simulator. The most reliable simulation can only be performed after porting the compiled VHDL into the implementation software. Simulinkdoes not:

- Do fixed point arithmetic in the general sense (expected in a future version).
- Have data types compatible with bit logic (bits can only be simulated with floating-point).
- Incorporate propagation delay in its blocks, which is not relevant at this level of abstract, but necessary for the implementation.
- Support reusable symbols (they may have different contents and the same name).

In the structural simulation using bit logic arithmetic it is possible to force Simulink™ to assign only 0s and 1s, even though they are represented with floating-point variables/signals. Fixed-point arithmetic can be implemented structurally in Simulink™ using gates. This

also simplifies setting propagation delay, as this could be included into the VHDL description of each gate. However, this is not possible in the Simulink™ model. Summarising, the structural fixed-point design can be quite easily converted into VHDL directly, without much additional intelligence required from the conversion program.

The model description of the Simulink™ block (MDL-file) is very similar to the representation of the common structure. It contains both the parameters of the simulation, description of each block with parameters for each block and block connections. The problem is that Simulink™ does not use reusable symbols. This means that if there are several blocks or symbols of the same name, they are all fully duplicated to the most basic element. These makes the analysis of common blocks much more difficult as these blocks may have slight differences and then qualify as two different ones, even if they have the same name. Therefore, the designer must obey the rule that all blocks having the same symbol must also have the same contents. They may only have different parameters.

2.3. Structural Analysis of the Simulink Model Description.

As it was pointed out earlier, the description of the Simulink™ model has close resemblance to the Matlab™ structure definition. Describing the model with the structure would allow simplifying the conversion process as inter-dependence of blocks could be indicated by their position in the tree of blocks. Therefore the conversion of the MDL-file into the Matlab™ structure was the first task to be done by the conversion utility developed.

The main problems faced in this stage were:

- The structure obviously can not allow the same field names at the same level, which was allowed in the MDL-file. All the blocks and lines (connection signals) had to be renamed consecutively as a remedy to this problem. Alternatively they can be combined into a vector.
- There are no commas to separate parameters and values in the MDL-file, required by the structure syntax. They had to be included appropriately.
- There is an inconsistency in the description of text constants. In Matlab™ they are indicated by a single quote, in the MDL-file by the double quote. Therefore single quotes were replaced by double quotes wherever the text constant was found.
- Simulink™ does not require ports to have their width always defined. This created confusion in specifying the number of input/output signals in the entity definition. The safest solution was to make a rule of explicitly defining the width of the ports in the Simulink™ model wherever it was possible. Even so there were cases when the data type had to be derived indirectly from the block to which the port was connected.
- The number of input and output ports was not defined consistently. For some Simulink™ blocks they were clearly given by the parameters "Inputs" and "Outputs". For other ones there was only one parameter "Ports", containing a five element vector with the number of input ports in the first element and output ports in the second element. There were also several blocks for which there was no description of the number of ports at all. For such a case whether the block had input or output port had to be derived from the connection description ("Line").

The main keyword in the MDL-file to look for is "System". This indicates the beginning of the description of the blocks and their connections within one block. It is then followed by a number of "Block" sections describing components of the design and "Line" sections each equivalent to a single wire connector (one can connect to multiple outputs). The "Block" can have another "System" section, which means it contains a lower-level circuit description. Sometimes such blocks also have some mask parameters. This indicates that there has been a symbol created for such a block. In this case "Mask type" describes the common symbol name (which could be used for the entity name later), "MaskPromptString" contains descriptions of the symbol parameters, "MaskInitialization" has their names and "MaskValueString" their values. If no "System" is found it means that the block is the basic component of the Simulink™ library and its description should be later copied from the library of basic VHDL blocks.

The "Line" statement contains the names of one source block and one or more output ones and their port numbers. For multiple output ports each of them is described by its own "Branch" statement.

There are also other block parameters like "Decimation" and "SamplingTime", which are useful for multirate systems. These have not been used in the current version of the MDL-to-VHDL conversion program.

2.4. Automated Conversion from Simulink to VHDL.

In order to simplify the first version of the conversion program, it has been designed with some constraint put on the original Simulink™ model. The model was required to:

- Operate on bit signals or vectors of bits
- Have only one sampling rate throughout the design
- Be composed of gates, constants, ports and buses only

This allowed the generation of the structural VHDL description relatively easily. The next versions of this

toolbox will allow different variable types and generate structural or behavioural VHDL wherever applicable.

The conversion requires two passes. First it looks through the whole design identifying common blocks of the model, each of which would be described in a separate VHDL file. It distinguishes the sub-blocks of the model from the basic Simulink™ blocks. It also gathers information about ports of each block and their types. This information is needed for creating "component" statements in the VHDL file.

At the second pass the algorithm looks recursively through the whole hierarchy of the model from the top level down to the bottom one creating the structural description of each block found in the first pass. For each of them it finds the list of "blocks" and the list of "lines". The first ones are used to generate block instantiation and configuration commands and the latter ones to define the internal signals. The entity definition is being created from the information found in the first pass of the conversion.

2.5. The Basics of the Polyphase IIR Half-Band Lowpass Filter Structure

The idea of converting the Simulink™ design into VHDL has been tested on the example of the two-path two-coefficient polyphase filter [5,6].

The basic recursive (IIR) allpass filter, shown in Figure 1(a) is the core of the polyphase IIR structure. For the case of the half-band lowpass filter two of such allpass blocks have to be used in a two-path configuration shown in Figure 1(b), with the appropriate delay in one of the branches. The higher order filters can be obtained by cascading a number of basic allpass filters in each path of the polyphase structure, taking care only to keep the number of them similar in both paths. By carefully designing the coefficients, the structure allows to obtain a very high-performance and relatively easily implementable half-band lowpass filter.

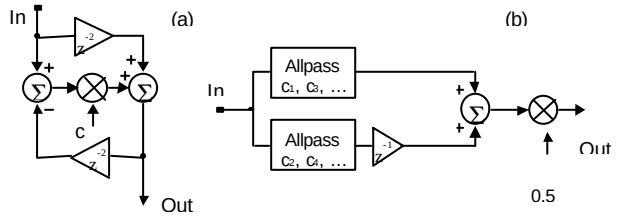

Figure 1. The N-D Form Allpass Filter Structure, (a), and the Two-Path Half-Band Polyphase IIR Lowpass Filter Structure, (b).

Design techniques for such polyphase half-band recursive IIR filters employing parallel/cascade combinations of elementary all-pass sections having one coefficient per second-order stage, as the starting point for an eventual elliptic approximation, have been reported in depth [5,6]. The algorithm for generating the prototype allpass filter coefficients for floating-point precision coefficients [1] is based on the analogy to elliptic filters. However, for effective real-time physical realizations (fixed-point) finite wordlength coefficients are required and need to be established, [7].

The basic building block from Figure 1(a) is the 2nd-order IIR allpass having its two poles on the imaginary axis and its two zeros on the same axis, but at the reciprocal distance from the origin. It has the transfer function of (1).

$$H(z) = \left(cz^2 + 1\right) \Big/ \left(z^2 + c\right) \tag{1}$$

There exists a variety of physical structures, which implement (1). The structure choice we have made here is that of the Numerator first, followed by the Denominator (N-D form), computations. By doing the calculations in this manner, relatively low peak gains at intermediate points in the structure are achieved, at a cost of the minimum number of computations. The physical structure of the basic Numerator-Denominator form (N-D) 2nd-order allpass filter is shown in Figure 1(a).

Configuring the appropriate order all-pass sections in a parallel fashion, with a delay in one of the branches, as shown in Figure 1(b), results in an overall lowpass half-band filter as shown in Figure 2.

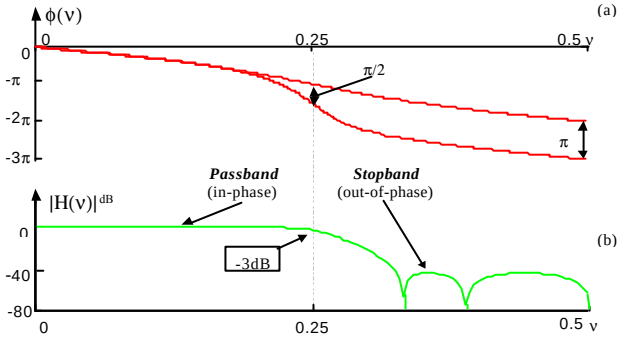

Figure 2. The influence of the phase responses of the allpass filters, (a), in both branches of the polyphase structure on the magnitude response of the overall lowpass filter, (b).

The effect on the pole zero pattern (PZP) is that its poles are at the same locations as for the allpass filters with an addition of an extra pole at z = 0 (due to the delay added to the lower branch). The zeros on the other hand are transported to new locations, with an additional zero introduced at the Nyquist frequency (Figure 3). For the case when second-order allpass sections are used, the magnitude response of each stage in Figure 2 is given by (2).

$$\left|H_C(z)\right| = \left|G(z)\left\{a_1(z+1) \Big/ z\left(z^2 + a_1\right)\left(z^2 + a_0\right)\right\}\right|$$
$$G(z) = z^4 + \left(\tfrac{a_0}{a_1} - 1\right)z^3 + \left(a_0 + \tfrac{1}{a_1} - \tfrac{a_0}{a_1} + 1\right)z^2 + \left(\tfrac{a_0}{a_1} - 1\right)z + 1 \tag{2}$$

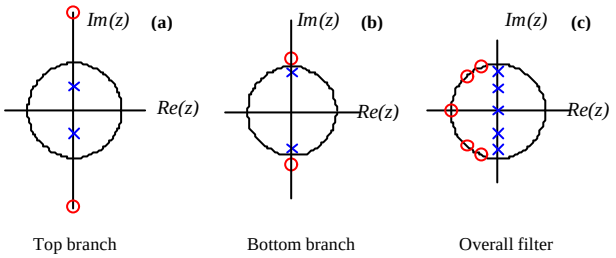

Figure 3. The PZP of allpass filters in both branches of the two-path polyphase structure, (a) and (b), and the PZP of the overall lowpass filter, (c).

The gain at dc (z=1) for this class of filter, is unity, with zero gain at Nyquist (z = -1), and is down by -3dB at half-Nyquist (z = j), irrespective of the filter coefficients, a_0, a_1, ... or the order of the allpass sections. The best way to explain the working of the filter is via phase responses of the allpass filters (since the magnitude is unity throughout). There exists a phase shift of exactly π (due to the unit delay in the bottom path) at Nyquist between the two branches, and that both branches are in phase at DC.

There is, however, a sharp transition in the phase at half-Nyquist as the poles on the imaginary axis are approached and passed while traversing the unit circle from DC to Nyquist. Hence the rationale here is that the (top and bottom branch) filter responses add constructively (as they are in phase) from dc to half-Nyquist forming the new filter's pass-band and add destructively (as they are π out of phase) from half-Nyquist to Nyquist, forming the new filter's stopband response. A simple yet very effective way of getting high levels of stopband attenuation, without substantially affecting the passband performance is through cascading of lower-order structures.

Since the transfer function of the half-band allpass filters involve only polynomials in z^2 , the polyphase structure incorporating them is very attractive for two-times decimation (interpolation) arrangements as the sample rate reduction (increase) can be moved to the input (output) of the polyphase filter. The unit delay in the lower branch is effected by feeding even samples into the top branch and odd samples into the lower branch (effectively performing undersampling) [7]-[8].

2.6. The Simulink Simulation Set-up

The example design is implementing a two-coefficient (c1=0.125 and c2=0.5625) polyphase lowpass filter as shown in Figure 1. The design was first captured using standard floating-point Simulink™ blocks. In order to make it close to the implementation the results of additions were rounded-to-zero to 20-bits (Function 1), subtractions truncated to 20-bits (Function 2) and multiplication truncated to 24-bits (Function 3). Local increase of wordlength at the multiplication was decided upon in order to avoid the unnecessary loss of precision before the subsequent addition. All data was being represented in two's complement arithmetic with 2 integer bits and a sign, which gives enough guard bits to deal with internal calculation, 20 altogether.

```
function [out]=rtz20(in)
fraction = 2^20; x=fraction.*in;
out=fix(x)./fraction;
```

Function 1 The 20-bit Round-to-Zero Function.

```
function [out]=t20(in)
fraction = 2^20; x=fraction.*in;
out=floor(x)./fraction;
```

Function 2 The 20-bit Truncation Function.

```
function [out]=t24(in)
fraction = 2^24; x   = fraction.*in;
out = floor(x)./fraction;
```

Function 3. The 24-bit Truncation Function.

Such a rounding scheme allowed eliminating of the limit cycles while keeping the DC offset low. The floating-point version of the filter has been compared to the architectural one designed from standard gates (Figure 4). The simulation uses a two-phase non-overlapping clock required by the delayors built from two D-type flip-flops per bit per unit delay. Flip-flops are active with the rising edge of the clock. The data is being read at the rising edge of *Clock1* and being available at the output at the rising edge of *Clock2*. The comparative simulation allows testing of the design for both an impulse and for the signal generated by the $\Sigma\Delta$ modulator. Results of both the fixed-point behavioural and the fixed-point structural versions of the design were exactly the same.

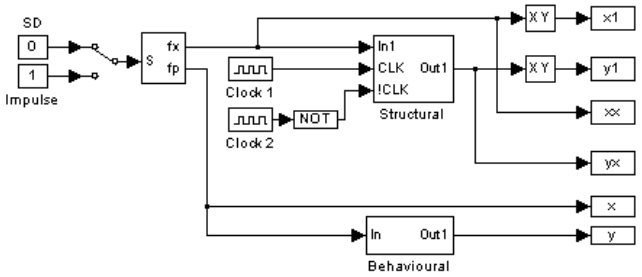

Figure 4 The test bench comparing fixed-point designs: the behavioural one and the structural one.

The fixed-point structural system has been designed to run from the external clock signal in order to be able to synchronise the filter with the input data for the ultimate physical implementation. The only blocks requiring the clock are the delayors, the rest is just combinational blocks for which the result is available at a certain time after the change of the input. This time is called the propagation time. The maximum propagation time is dependent on the propagation time of the gates and the maximum number of dependent gates the signal has to go through.

Figure 5 shows the inside of the fixed-point polyphase lowpass filter and Figure 6 describes the allpass structure used for both the *UpperBranch* and the *LowerBranch* blocks (the only difference being the multiplication factor). The floating-point design looks very similar to the fixed-point one, it just does not have the clock signals since Simulink™ itself is controlling the simulation.

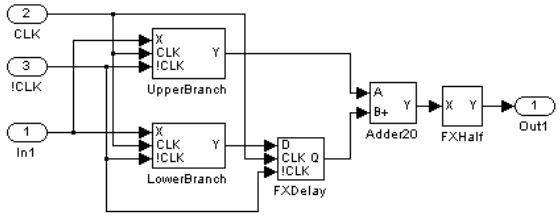

Figure 5 The fixed-point half-band filter structure.

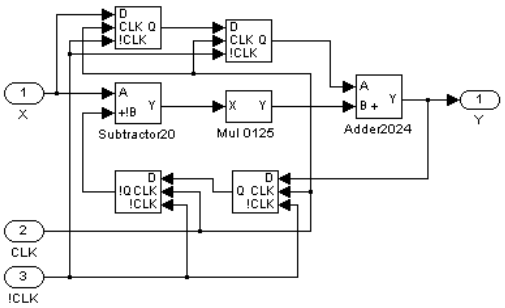

Figure 6. The fixed-point 2nd-order allpass filter structure.

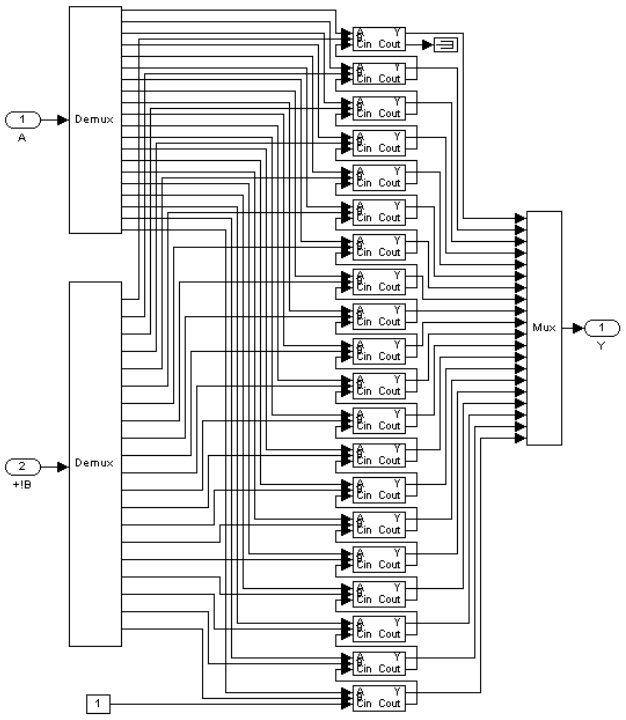

Figure 7. The fixed-point 20-bit subtractor structure.

The structure of the 20-bit subtractor with truncation is shown in Figure 7. The second input is being negated (taking the negated output from the delay block) and increased by one in order to inverse the sign before being added to the first input. As the two's complement arithmetic is used, negation is achieved by inverting all the bits at the delayor (Q! output) and adding one using a ladder of two-bit adders with carry input (Figure 8). Assuming the same propagation delay for all gates, T_P, the maximum time required to add two numbers is $20T_P$.

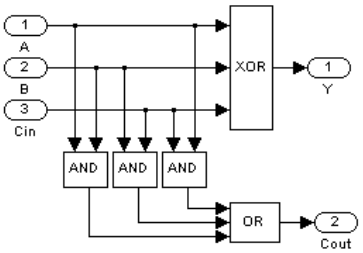

Figure 8. The two-bit adder structure with carry input.

The 20-bit wide delayors in Figure 9 have been designed using two D-type flip-flops in the Master-Slave arrangement for each bit. The *Mux* and *Demux* are just converting the single bit lines into the vector of bits and back again. They were used for the purpose of the simulation only and were not to be implemented.

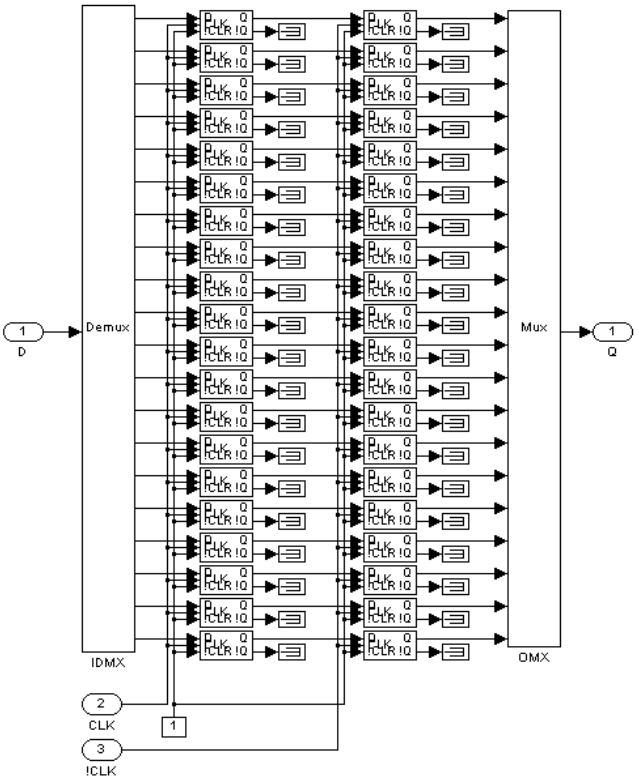

Figure 9. The fixed-point 20-bit delayor structure.

Notice the termination blocks (new feature in Simulink™ 3) used to avoid warnings about unconnected outputs. The multiplication by 0.125, required in the *UpperBranch*, effectively means shifting data three bits towards the Least Significant Bit (LSB) as in Figure 10. In order to take care of the negative numbers in two's complement arithmetic, the Most Significant Bit (MSB) has been propagated to the next three bits (sign extension). The output is given in 24-bits without any loss of precision. Actually, 23-bits is enough to provide the full accuracy. However, 24-bits sizing have been chosen for the consistency with the other multiplier by the factor of 0.5625.

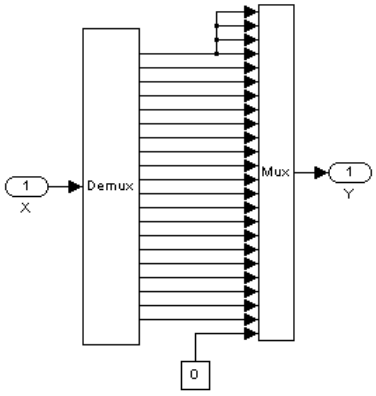

Figure 10. The fixed-point 14-bit 0.125 gain structure.

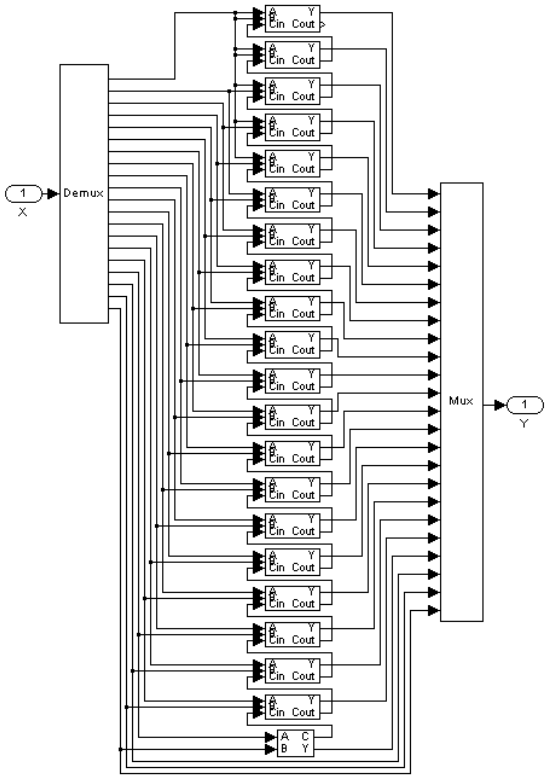

Figure 11. The fixed-point 0.5625 gain structure.

The multiplication by 0.5625 is a bit more complicated (Figure 11), as this requires adding together two shifted versions of the input, by one bit (0.5 factor) and by four bits (0.0625 factor). For such a case 24-bits are required to provide the output at the full accuracy. The result is available after a maximum time of $20T_P$.

The simplest case of the multiplier is the half divider (Figure 12) required at the output of the filter to scale the transfer function to unity for low frequencies. This has been achieved by one bit right shifting of the input signal. The result has been truncated to 20-bits by disregarding the LSB of the input data.

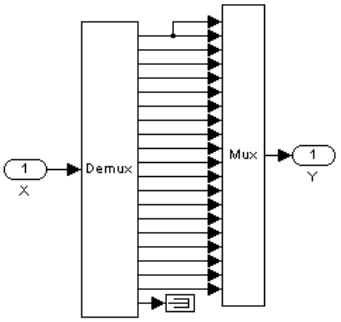

Figure 12 The fixed-point one-bit right shifter structure.

The result of the multiplication by 0.125 or 0.5625 is being added to the delayed samples of the input in the adder, the structure of which is shown in Figure 13.

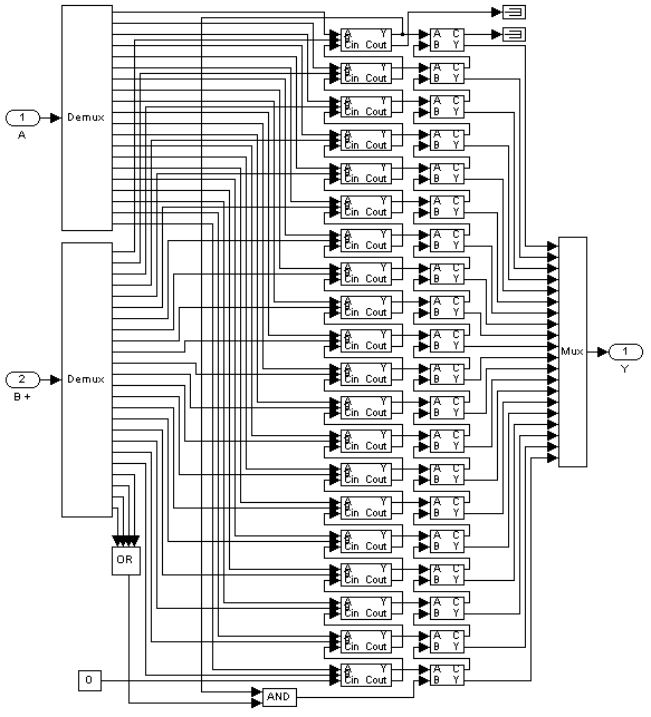

Figure 13. Fixed-point adder structure (20-bit + 24-bit) with the 20-bit result rounded to zero.

The result of adding a 20-bit input to the 24-bit one is subsequently constrained back to 20-bits using a round-to-zero scheme. This is achieved by the OR and AND gates. The four-port OR gate examines if there is any 1's set among the disregarded bits. If the MSB=0 (positive number), the output of the OR gate is disregarded forcing to truncate the data. If the MSB=1 (negative number), the result of the OR gate is added to the output rounding it up towards zero. The maximum propagation time of the block is $35T_P$. The adder from Figure 13 required a two-bit adder with no carry input, which is shown in Figure 14.

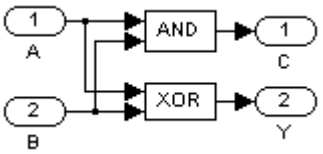

Figure 14. The two-bit adder structure without carry input.

The 20-bit addition with truncation of the result has been implemented as in Figure 15. No loss of precision happens here as the format of the data is such that the possibility of a carry bit set does not influence the performance of the structure. The maximum propagation time is $34T_P$.

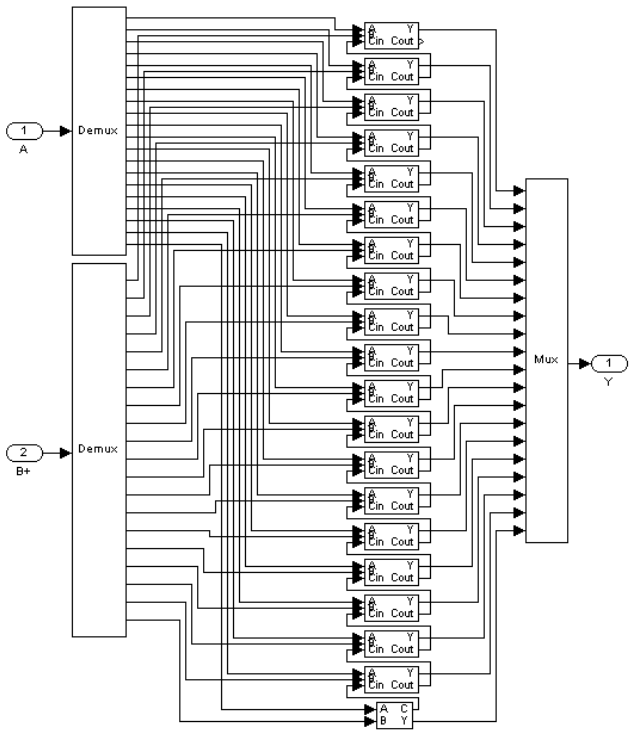

Figure 15. The fixed-point 14-bit adder structure.

The result of the Simulink™ simulation showing the magnitude response of the implemented filter with its stopband attenuation is given in Figure 16. The shape is very close to the one, which can be obtained from the floating-point simulation. The same magnitude response zoomed into the passband showing the ripple structure is presented in Figure 17. For the comparison, the floating-point simulation result is shown in dashed line. As it can be expected, the latter one shows much smaller ripples.

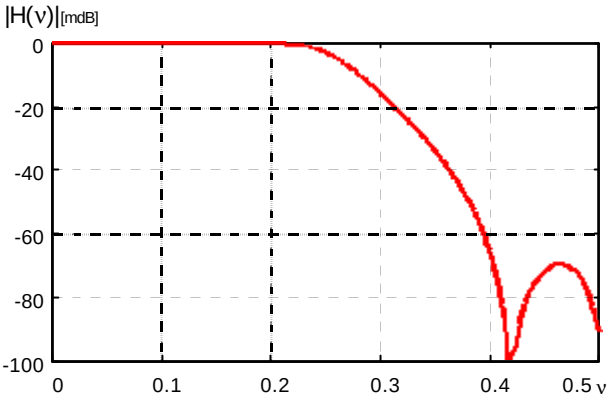

Figure 16. Magnitude response of the implemented polyphase halfband lowpass filter showing the stopband attenuation.

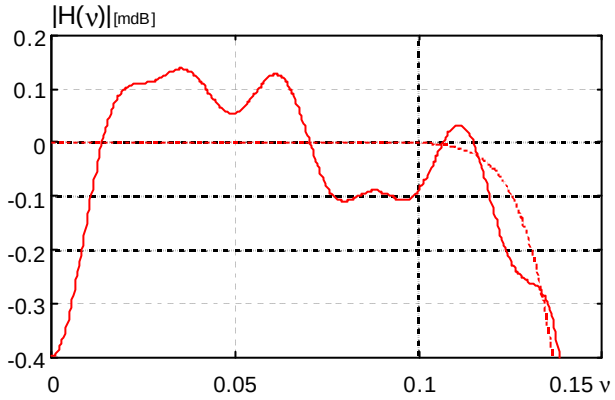

Figure 17. The passband ripples of the implemented polyphase halfband lowpass filter (solid line) and the theoretical one (dashed line).

The polyphase IIR structure with its high performance and low sensitivity to coefficient quantisation and use of a small number of coefficients, has been found very attractive for decimation filters (the core of which is the lowpass filter) for Sigma-Delta (ΣΔ) based Analog-to-Digital Converters (ADC). It is required for the decimation filter to work very fast as the rate of the data coming from the ΣΔ modulator is many times higher than the one required by the sampling theorem to which it is sampled down after the lowpass filtering. It is shown in Figure 18 the example spectrum of the output of the ΣΔ modulator excited with single sine input and the output of the polyphase filter. The quantisation noise at high frequencies is very clearly decreased while the signal at low frequencies is practically not affected at all.

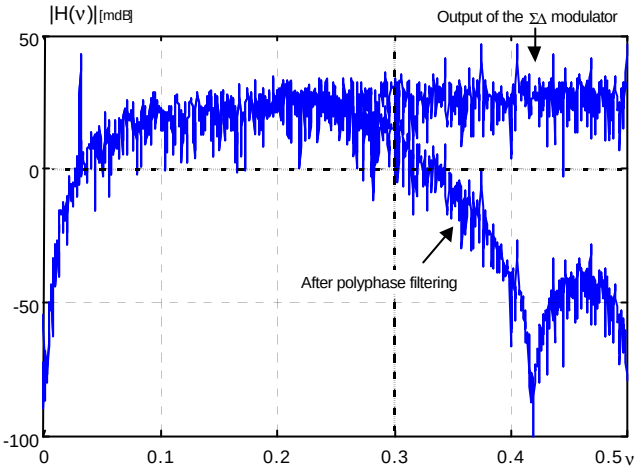

Figure 18. The example spectrum of the output of the ΣΔ modulator excited with single sine input and the output of the polyphase filter.

The size of the datapaths has a direct influence on both the filter stopband attenuation and its passband ripples. At least 14-bit datapath was required for the attenuation to fall down below 60dB. A minimum of 20-bit datapaths was required in order to achieve less than 0.5mdB peak-to-peak passband ripples.

2.7. Conversion of the Model to VHDL

The conversion of the MDL model into VHDL has been done with the preliminary version of the custom Matlab™ program. Because of the difficulties in the analysis of the Simulink™ description the resulting VHDL code required some additional editing. The basic blocks like D-type flip-flops with reset, standard logic gates and the two-phase clock have been designed manually in behavioural description as part of the library of standard blocks.

A separate test bench has been written. Its purpose was to compare the results of the VHDL simulation with the output from the fixed-point Simulink™ model. The complete output of Simulink™ run has been stored in the file comprising all bits of the input and the output. This file has been read sample-by-sample and compared with the output of the VHDL simulation at each clock cycle. The compilation and simulation of the VHDL code, and subsequently its synthesis, has been done with PeakFPGA from Accolade Design Automation Inc. [3] provided by the company for the purpose of evaluation. The example screen shot of the simulator software running the designed VHDL code is shown in Figure 19.

The VHDL simulation has some differences from the high-level Simulink™ one:

- There is a need to take proper care of avoiding unassigned states by properly resetting the design before it starts operating. This has been achieved by setting a CLR_L to zero for the first 60μs of the simulation, before the first rising edge of Clk_1, i.e. first reading of the data from the input.
- The propagation time through the blocks of the design plays an important role in the design. This parameter is not considered in Simulink™ at all. The VHDL simulation allowed assessment of the maximum speed of operation of the design that was approximately four times the maximum settling time of the combinational logic. For the simulation provided here the clocking speed has been set to 3.3MHz, assuming 2ns propagation time for the logic gates.
- The simulation in Simulink™ required only that Clk_1 and Clk_2 were non-overlapping clock signals. The VHDL simulation proved that the best performance (highest speed of operation) has been achieved when overlapping time was a quarter of the clock period.
- The test bench may be converted from Simulink™, but it is better to create a new one, which would compare the results from Simulink™ with the results of the VHDL simulation, exactly the way it was done for the two-coefficient (α_1=0.125, α_2=0.5625) example design.

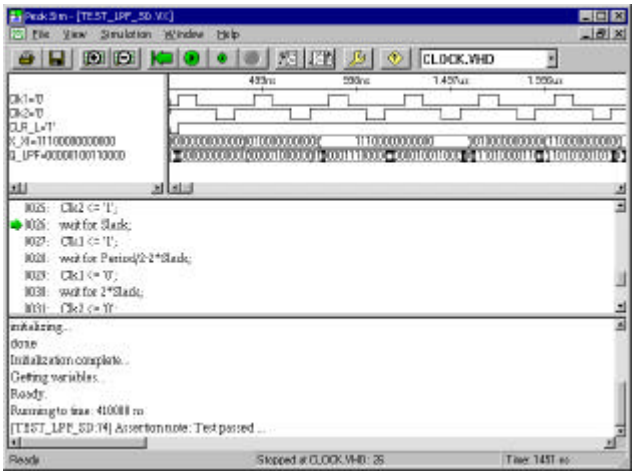

Figure 19. Screen shot of the simulation in PeakFPGA.

Technology	Area [FG]	Delay [ns]	DFlops (DEFs)	PI	PO	Instances
X-3000	264	248	240	23	20	550
X-3100	264	122	240	23	20	550
X-4000	252	350	240	23	20	560
X-5200	267	325	240	23	20	557
X-7200	1224	N/A	-	23	20	1755
X-9500XL	1454	N/A	240	23	20	1681

Table 1 Results of the Synthesis with Galileo for Xilinx.

The VHDL code designed for the example single-stage polyphase filter has been subject to synthesis in both PeakFPGA version 4.25 and in Galileo for Xilinx from Mentor [4]. The first one returned the same result for all design families, including Actel, Altera, Lattice, Lucent and

QuickLogic EDIF devices. It turned out that only 360 D-type flip-flops and 1132 two-input gates (677 LUTs) were required to implement the design (excluding the clock) generator.

The results of the synthesis by Galileo for Xilinx only were different for each design component. These results are presented in Table 1, showing the area, total propagation delay, number of D-type flip-flops, input pins, PI, output pins, PO, and instances. Basically Galileo calculated that only 240 flip-flops were needed for the delayors. The difference was in the number of gates required, between 267 and 1454 depending on the technology. Galileo, in contrast to PeakFPGA, also gave the estimated input-to-output delay between 122ns and 350ns, dependent on the technology used. The maximum clock frequency of the filter may therefore range from 1.1MHz for Xilinx-5200, 1.4MHz for Xilinx-3000 up to 2.9MHz for Xilinx-3100. It is dependent on the propagation delay of combinational logic, which is maximum 4.5ns for Xilinx-5200, 3ns for Xilinx-3000 and 1.5ns for Xilinx-3100. The propagation delay for Xilinx-9500XL is 4-6ns. The sequential delay of the flip-flop is merely up to 6ns and all of them work in parallel. Therefore the preferable technology to implement the filter could be Xilinx-3100A, giving the best speed of operation at low cost and optimum use of the FPGA. Assuming 0.1µm technology with a transistor size of 0.25µm, the gate consisting of four transistors and each flip-flop consisting of eight the estimated total size of the components of the design would be approximately 0.2mm by 0.2mm plus few percent for the connections.

3. Conclusions

The example design of the polyphase filter and then its conversion into VHDL proved that such an idea would be a very attractive way of designing test chips very quickly. It took three days to get from the Simulink™ model to its final synthesised version. The next stage of the research work would be either to compile to a custom layout and put it onto silicon or to commit the design onto a standard FPGA.

The current version of the program performs only direct mapping of structures from Simulink™ to VHDL and does not work for multiplexed architectures. In order to do such conversion the program requires an algorithm analysing behavioural or structural descriptions to find common operators, and convert them into the multiplexed structure with added control circuitry. This will be the aim of the future work.

The DSP group in the Department of Electronic Systems has been active in the area of polyphase filters and their application for decimation and interpolation structures for ADDA for a number of years. The polyphase structure proved itself to be the best to be implemented in the very fast and high accuracy decimators. The specimen filter, presented in this paper, could be comfortably used for the first four stages of the decimation filter described in [5]. Even when considering that the design has to be repeated eight times, the total required silicon area of 0.5mm by 0.5mm is very tiny. Putting together the hardwired filters would avoid the need for fast processors, giving more space for the analogue part of the A/D converter, hopefully the whole ΣΔ modulator. The small size implications are a big advantage as this would free up silicon real estate for the implementation of other functions.

Acknowledgements

Authors would like to thank Accolade Design Automation, Inc. for allowing to use of the evaluation version of the Peak VHDL package, used here to verify the results of the Simulink-to-VHDL conversion.

References

[1] Ashenden, P. J., *VHDL Cook-Book*, First Edition, Department of Computer Science, University of Adelaide, South Australia, July 1990.

[2] Holmes, C., *VHDL Language Course*, Rutherford Appleton Laboratory, Microelectronics Support Centre, Chilton, Didcot, 23-25 May 1995.

[3] Accolade Design Automation, Inc. WEB site: http://www.acc-eda.com

[4] Mentor Graphics WEB site: http://www.mentor.com/products/alphaindex.html

[5] harris, f., "On the design and performance of efficient and novel filter structures using recursive allpass filters", *Proceedings 3rd ISSPA 92*, vol. 1, pp. 1-5, Gold Coast, Queensland, 16-21 August 1992.

[6] harris, f., M. d'Oreye de Lantremange and A. G. Constantinides, "Digital signal processing with efficient polyphase recursive all-pass filters", *International Conference on Signal Processing*, Florence, 4-6 September 1991.

[7] Kale, I., R. C. S. Morling and A. Krukowski, "A high-fidelity decimator chip for the measurement of Sigma-Delta modulator performance", *IEEE Transactions on Instrumentation and Measurement*, vol. 44, no. 5, October 1995.

[8] Krukowski, A., I. Kale, K. Hejn and G. D. Cain, "A bit-flipping approach to multistage two-path decimation filter design", *Second International Symposium on DSP for Communication Systems*, SPRI, 26-29 April 1994.

Address for correspondence.

Dr. Artur Krukowski
University of Westminster,
Department of Electronic Systems,
London W1M 8JS, United Kingdom.
E-mail: krukowa@wmin.ac.uk
WWW: http://www.cmsa.wmin.ac.uk/~artur