

Utilization of a VHDL-based ASIC-Realizable Digital Filter Architecture Library in DSP System Design

Vesa Savela, Petri Järvinen, Arto Nummela, Jari Keskinen, Jari Nurmi

Tampere University of Technology
Signal Processing Laboratory
P.O. Box 553, FIN-33101 Tampere, Finland
Email: {vjs,pja,nummela,kessi,nurmi}@cs.tut.fi

Abstract

This paper reviews the different implementation architectures of several digital filters, using synthesizable VHDL. The key issue on developing functions with VHDL descriptions is that VHDL allows usage of a number of parameters while developing DSP functions, but it also requires an optimized VHDL coding style for a specific synthesis tool. A recently created VHDL macro function library provides different kinds of realizable filter architectures for a variety of DSP applications. The library functions are parameterizable and generic. The library development started from the industry requirements to increase the design efficiency of ASICs, and to enable a rapid evaluation of various implementation architectures.

1. Introduction

The focus of DSP system development has recently moved from gate-level optimization towards higher level optimization and specialist synthesis tools for certain common architectures. Basic motivations for using these modern tools are to increase productivity, to shorten the evaluation time of the optimized design and to standardize the descriptions, to minimize the silicon area utilization, to get first-time-correct circuits, and to manage complex designs [1][4].

The recent trend is to move towards more efficient reusability by developing user or application specific macro function libraries, such as parameterizable high-level DSP models in

VHDL. The macro function library consists of optimized complex arithmetic blocks and DSP filter architectures, allowing more time for specifying the system and making technology-independent architecture selections. Even a small subset of a parameterized macro function library can cover a number of different DSP applications [2] [8].

We have utilized several different filter architectures to find suitable ones for various applications. These filters are gathered into a DSP macro function library. We have used modular design style with VHDL and synthesis tools to generate rapidly application specific variations of macro functions and to improve the reusability and design efficiency. Using these library components it is easy and fast to evaluate the final required silicon area and performance of the components [4].

Each of the macro functions is accompanied by a data book, since the commenting in the VHDL code is not enough to fully specify the features of the resulting hardware. In the data book there are examples of the speed and complexity of the block using certain parameter values and implementation technologies. Also the underlying theory can be better enlightened in the data sheets.

2. Reusable and parameterizable DSP library

The function library includes entities such as FIR, IIR and adaptive filter blocks, which can be

used as parameterizable macros. The parameters include, e.g., number of coefficients, symmetry, coefficient values, internal accuracy, and input data, output data and coefficient word lengths for each of the filter architecture.

The parameters of a filter are passed into the actual instantiation through a package file containing also the filter coefficients. An example is shown in Figure 1. This package information of the file is then used in the VHDL code of the filter.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

package CoefPkg is
  constant Symmetry : std_logic := '1';
  constant AntiSymm : std_logic := '0';
  constant Taps      : NATURAL := 16;
  constant CWidth   : NATURAL := 12;
  constant DWidth   : NATURAL := 12;
  constant IntAcc    : NATURAL := 13;
  constant Outbits  : NATURAL := 12;
  type CoefArray is array (0 to Taps-
    1) of signed(CWidth-1 downto 0);
  constant coef: CoefArray :=
("11111111101", -- b(0) = -1.453696e-03
"00000000001", -- b(1) = 5.148491e-04
...
"00000000001", -- b(14) = 5.148491e-04
"11111111101"  -- b(15) = -1.453696e-03
);
end CoefPkg;

```

Fig. 1. An example of a package file

In this paper the main FIR and IIR filter implementation architectures are reviewed. The achievable sampling rate and gate count with different parameters are reported. The comparison information of the main properties is provided by using synthesis results. Based on these comparison results, the DSP implementation groups can select the most suitable architecture for a certain application.

The main emphasis is on showing the versatility of a parameterized macro function library in DSP system development. The system or DSP engineers can rapidly evaluate the different hardware alternatives on silicon. The design

space can be explored within a working day instead of months, without any ASIC expertise.

The main application area of the DSP macro cell library is in our case the signal processing algorithms needed in the design and implementation of ASICs for telecommunication applications.

3. Requirements of the DSP architecture

Typically, DSP algorithms are very repetitive, for example FIR filters, or have many things to be calculated, like in the case of an IIR biquad filter. The following list includes the typical requirements of DSP library functions:

- fast sampling cycle - this is different from high system clock speed
- cycle time adjustable according to the working environment
- fast math operations (efficient /small hardware multiplier, adder)
- simple arithmetic blocks - fixed point arithmetic - saturation logic available
- possibility to utilize external or on-chip RAMs
- flexible bussing capabilities
- duplicate resources for parallel computation of the real and imaginary components of complex numbers
- hardwired control for failure protection
- inexpensive and easy-to-develop peripherals
- lower power consumption with a standby mode, and minimal number of state changes.
- fast and reliable, easily convertible and up-graded
- programmability

4. Filter architectures

The two basic digital filters are FIR (Finite Impulse Response) and IIR (Infinite Impulse Response) filters. The theory is well-known and good software exists for designing and optimizing the filter coefficients. There are many other filters and other DSP algorithms, of which some are included in our macro library. They are usu-

ally not so often used and only one implementation is sufficient.

4.1. Finite Impulse Response (FIR) filters

Each type of digital filter architectures has its advantages and disadvantages. An FIR filter is always stable because there is no feedback from the output and thus the impulse response is finite in time. In addition, the amplitude and phase can be arbitrarily specified. On the other hand, an FIR filter will generally require more taps, and consequently more math, to compute the output value.

The finite impulse response filter algorithm is a representative of a number of DSP equations found in convolution, filtering, and modelling. The requirements are simple but varying. The algorithm is multiply-addition intensive and has a simple but long loop characteristic. The system transfer function of the FIR filter is

$$H(z) = \sum_{k=0}^N b_k z^{-k}$$

The actual hardware implementation can be done in many ways. Parallelism can be used to maximize speed, or the architecture can be multiplexed to minimize the silicon area. Calculations, i.e. multiplications and additions can be implemented in several ways, e.g. using a parallel or a serial algorithm and the precision can be different for each multiplication, etc.

4.1.1. Fixed parallel architectures

The filters with a parallel architecture are suitable for low-order, high performance applications. Their main disadvantage is the relatively large die area consumption. Total parallelism requires duplication of all the arithmetic blocks (multipliers and adders).

The values of the coefficients affect heavily the synthesis result. The best result is achieved when the binary representation of the coefficient has most bits zeroes and only few ones. For example, when the coefficient has only one '1' bit, the multiplier is implemented as a simple shifter which does not need any gates in a parallel architecture. Selecting optimum negative

coefficients is not so easy and depends also much on the optimizing algorithm of the synthesis software.

We have developed code for two separate fixed coefficient parallel FIR filters: a direct and a transposed fixed FIR-filter using the parallel dataflow architecture, and these functions are suitable for both odd and even order filters. Additionally (anti)symmetric coefficients can be used to reduce the number of multipliers.

4.1.1.1. Direct form

The dataflow representation of the direct FIR is shown in Figure 2. [3][5] The number of implemented multipliers is only half of the number of taps when the coefficients are (anti)symmetric. In that case extra adders or subtractors are needed before the multipliers. Due to parallelism the gate count can be very high when implementing high order filters.

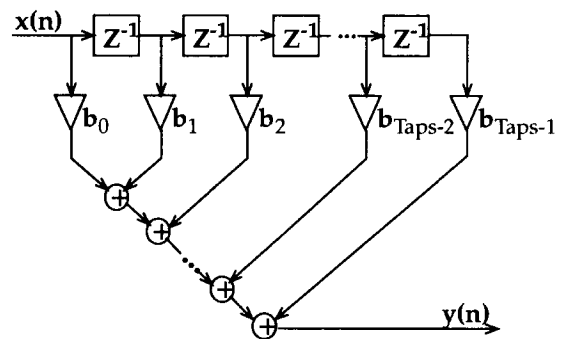


Fig. 2. Direct form FIR, dataflow representation.

4.1.1.2. Transposed form

The dataflow description of the transposed form FIR filter is shown in Figure 3. [3][5] The symmetrical coefficients are taken into account automatically by synthesis software unlike in the direct form. Calculating output $y(n)$ needs only one multiplication and one addition, so the maximum speed is very high. The weak point is that the registers must be bigger than in the direct form and typically the gate count is much higher except when coefficients are appropriately chosen.

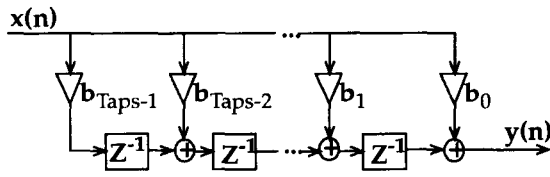


Fig. 3. A transposed FIR.

4.1.2. Programmable parallel architectures

The other possibility of implementing filters is that the coefficients are not fixed but are loaded one by one to the filter during run-time instead. The multipliers and adders inside the filter cannot be optimized and the gate count is very high. An additional register bank for storing the coefficient values is needed.

4.1.2.1. Programmable transposed form

The only parallel programmable FIR filter implemented for the macro library is of transposed form. The dataflow representation is similar as the fixed model shown in Figure 3. Coefficient values are loaded to registers from input port when enabled with appropriate control signal.

4.1.3. Programmable multiplexed architecture

The implementation of the programmable multiplexed FIR filter is based on a dataflow architecture using only one multiplier and an adder constituting a multiply- and accumulate (MAC) unit for arithmetic operations. The operands are stored in appropriate synchronous register banks. The block diagram of the implementation architecture is shown in Figure 4. The adder following the delay line is implemented only if coefficients are symmetrical and it is substituted by a subtraction unit if the coefficients are antisymmetric. The multiplexer preceding the multiplier will be implemented only if the filter symmetry is odd.

4.1.4. Serial arithmetic

All filters described in the previous chapters used parallel arithmetic. Another possibility is to use serial arithmetic instead. When using serial arithmetic, multipliers can be implemented

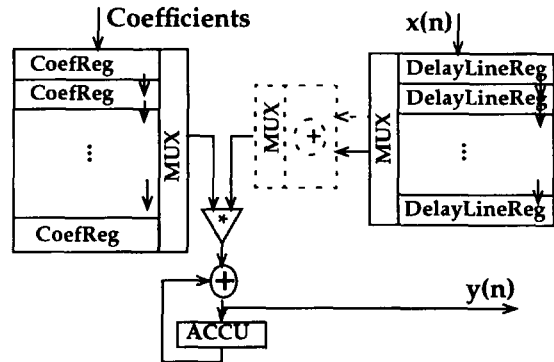


Fig. 4. Multiplexed dataflow FIR filter using one MAC.

using fewer gates. The detailed implementation of a serial arithmetic FIR filter is quite complicated and out of the scope of this paper. The FIR filter included in our library consists of five entities and uses parallel-serial arithmetic for multiplications. The overall dataflow structure is similar to the parallel dataflow implementation shown in Figure 4.

4.2. Infinite Impulse Response (IIR) filters

An IIR will generally have fewer coefficients, but the required output feedback can make the circuit implementation more complex. A stable IIR filter can become unstable if the coefficients are not chosen properly to avoid digital math errors. The system transfer function of the IIR filter is,

$$H(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}}$$

High order IIR filters are not commonly used because of the possibility of overflow and quantization noise which can be minimized by cascading lower order IIR filters and distributing the overall gain for each IIR block [5].

4.2.1. Fixed parallel first or second order IIR

The most straightforward implementation is a

parallel structure with fixed coefficients. There are several possibilities to implement the structure. One of them is the transposed direct form II [3][5] shown in Figure 5. The macro in our function library can be used as a symmetric second order ($b_0 = b_2$) or first order IIR filter.

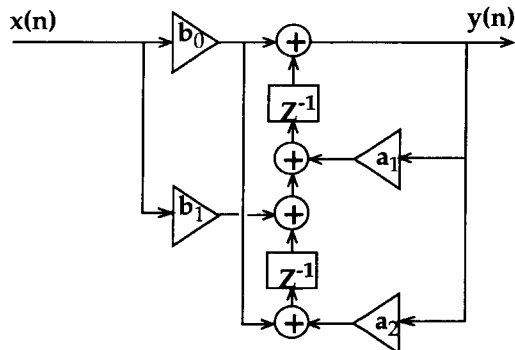


Fig. 5. Second order IIR, parallel architecture.

4.2.2. Programmable parallel first or second order IIR

The dataflow representation is similar to the fixed coefficient implementation but the coefficients are loaded one by one to the register bank during run-time.

4.2.3. Programmable multiplexed first or second order IIR

The implementation of the programmable multiplexed IIR filter is based on a dataflow architecture using only one multiplier and an adder constituting a multiply-and-accumulate (MAC) unit for arithmetic operations. The operands are stored in appropriate synchronous register banks. The dataflow representation of the implementation architecture for the second order IIR filter is shown in the Figure 6.

4.2.4. Programmable multiplexed second order IIR filter cascade chain.

The variation of the multiplexed architecture is a structure which implements a cascade chain of several second order IIR filters. One possible implementation is like the second order multiplexed architecture but the delay register substituted by a register bank. The dataflow

representation is shown in Figure 7.

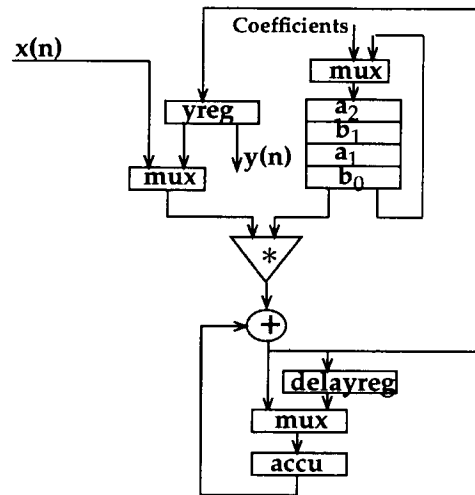


Fig. 6. Second order IIR, multiplexed architecture.

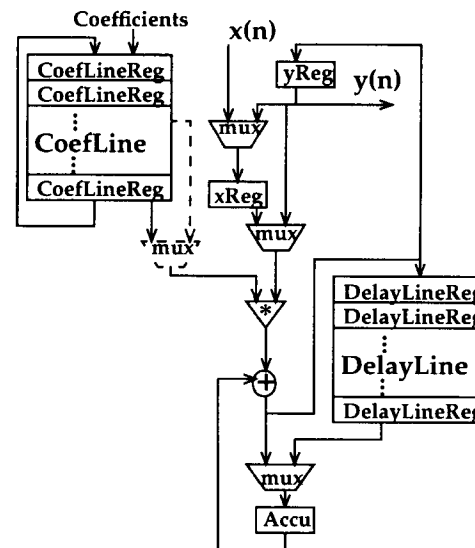


Fig. 7. Second order IIR cascaded chain, serial dataflow architecture.

The advantage of this implementation is that high order IIR filters can be realized using only one MAC with the better characteristics of the second order IIR filters. The implementation could also be done using arbitrary order IIR filters [7].

4.2.5. Serial arithmetic

The implementation using serial arithmetic is similar to the programmable multiplexed IIR (dataflow representation) and programmable serial FIR (arithmetic).

5. VHDL coding style

All macros written for the library are optimized for synthesis and commented. Testbenches for verifying the macro are also included. Additionally for each macro there exists a data book where the behavior and usage of the macro is explained as only commenting of the code is not enough. The library development is done with co-operation with several companies which all have their own guidelines and therefore one problem has been that our guidelines for writing VHDL code have evolved much during the project. The typical entity declaration for a filter is shown in Figure 6. Programmable implementations have also CoefIn and CoefLoad input ports and multiplexed implementations YRdy output port.

```
entity FirDirect is
  port (Clk : in std_logic;
        Rst : in std_logic;
        Ena : in std_logic;
        X   : in std_logic_vector
              (DWidth-1 downto 0);
        Of1 : out std_logic;
        Uf1 : out std_logic;
        Y   : out std_logic_vector
              (Outbits-1 downto 0));
end FirDirect;
```

Fig. 8. The entity declaration of 'FirDirect'

The basic structure of each entity is very similar. Two processes are used, one for combinational logic and one for synchronous logic.

An example of the process with combinational logic is shown in Figure 6. Several nested if-loops are needed so that the same entity can be used for different types of coefficient symmetries and counts by changing only the parameter package.

```
combin: process (XReg, ShiftReg)
... (variable definitions) ...

begin
  if symmetry = '1' then
    for i in 0 to HalfTaps loop
      if i = 0 then
        ExpNet(i)(DWidth) := XReg(DWidth-1);
        ExpNet(i)(DWidth-1 downto 0) := XReg;
      else
        ExpNet(i)(DWidth)
          := ShiftReg(i)(DWidth-1);
        ExpNet(i)(DWidth-1 downto 0)
          := ShiftReg(i);
      end if;
    end loop;

    for i in 0 to HalfTaps loop
      if Taps mod 2 = 1 then
        if i = HalfTaps then
          SumNet(i) := ExpNet(i);
        else
          SumNet(i) := ExpNet(i)
            + ShiftReg(Taps-1-i);
        end if;
      else
        SumNet(i) := ExpNet(i)
          + ShiftReg(Taps-1-i);
      end if;
    end loop;
    for i in 0 to HalfTaps loop
      MulNet(i) := Coef(i) * SumNet(i);
    end loop;
    for i in 0 to HalfTaps loop
      for j in DWidth+CWidth-1 downto 0
      loop
        Net(i) := MulNet(i)
          (DWidth+CWidth-1 downto 0);
      end loop;
    end loop;

    TmpIntAcc := (others => '0');
    for i in 0 to QuaterTaps loop
      if i = QuaterTaps then
... else ... end if;

    overf := (NOT TmpIntAcc(IntAcc-1)) AND
              TmpIntAcc(IntAcc-2);
    underf := TmpIntAcc(IntAcc-1) AND
              (NOT TmpIntAcc(IntAcc-2));
    Of1 <= overf;
    Uf1 <= underf;
    Y <= conv_std_logic_vector((saturate((
      TmpIntAcc(IntAcc-2 downto IntAcc-1-
      outbits)), overf, underf)), outbits);
  end process combin;
```

Fig. 9. The combinational process of the entity 'FirDirect'

The process with synchronous logic is shown in Figure 6. In this case the process is very short but the state machine for multiplexed structures can be very big.

```

synchr: process(Clk,Rst)
begin

  if Rst = '1' then
    ShiftReg <= (others => (others =>
      '0'));
    XReg <= ( others => '0');
  elsif Clk'event and Clk = '1' then
    if Ena = '1' then
      for i in Taps-1 downto 1 loop
        if i > 1 then
          ShiftReg(i) <= ShiftReg(i-1);
        else
          ShiftReg(i) <= XReg;
        end if;
      end loop;
      XReg <= signed(X);
    else
      NULL;
    end if;
  end if;
end process synchr;
end RTL;

```

Fig. 10. The synchronous process of the entity 'FirDirect'

The VHDL code of 'FirDirect' has total of 390 lines of which 200 lines are blank or comment lines.

6. Synthesis results

The three most important constraints when selecting appropriate implementation architecture are speed, area and power consumption [6]. However not only the implementation but also the input signals affect the power consumption, and not many high level tools support the estimation of the power consumption. Because of this we have not done any calculations about needed power.

All the filter implementations have been synthesized without any constraints using the Synopsys "Design Analyzer" synthesis software and LSI_10K ASIC-library.

6.1. Synthesis of FIR implementations

For the fixed coefficient architectures the results are shown only for example purposes because the values of the coefficients affect the results. The filter used was equiripple lowpass filter with $0.2f_s$ as the passband edge and $0.35f_s$ as the stopband edge. In the use of the parallel filter macro functions , the number of coefficients needs consideration due to large area consumption.

The needed gate count depending on the number of coefficients are shown in Figure 11. for symmetrical and in Figure 12. for asymmetrical cases. The accuracy of the samples, coefficients and output is 8 bits and the internal accuracy is 16 bits. In figures 13. and 14., the maximum sampling frequencies are shown for the same filters with the same parameters and coefficients.

solid	programmable multiplexed
dash-dotted	progr. serial arithmetic
dashed	programmable transposed
solid - o	fixed transposed
solid - *	fixed direct

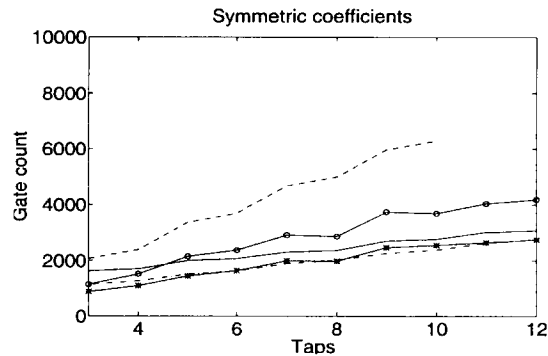


Fig. 11. Gate count as a function of the number of symmetric coefficients.

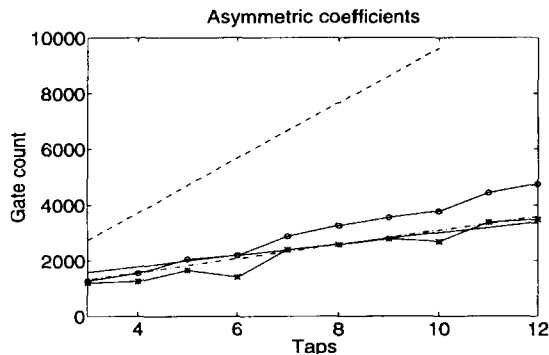


Fig. 12. Gate count as a function of the number of asymmetric coefficients.

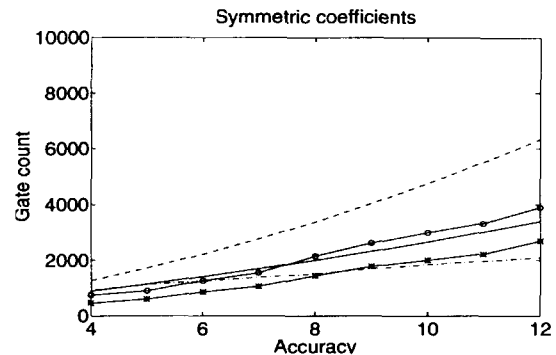


Fig. 15. Gate count as a function of the accuracy of samples and symmetric coefficients.

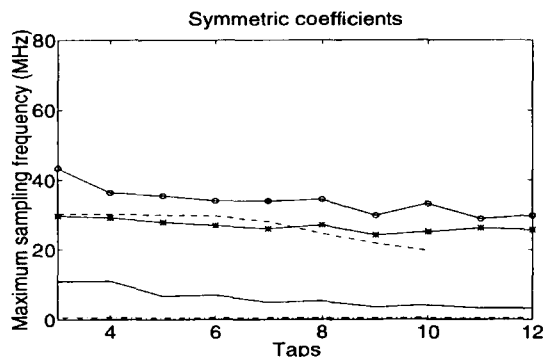


Fig. 13. Maximum sampling frequency as a function of symmetric coefficients.

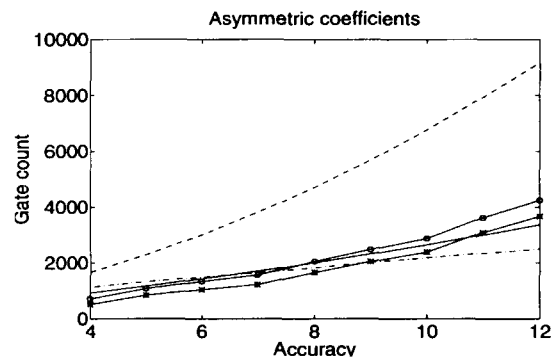


Fig. 16. Gate count as a function of the accuracy of samples and asymmetric coefficients.

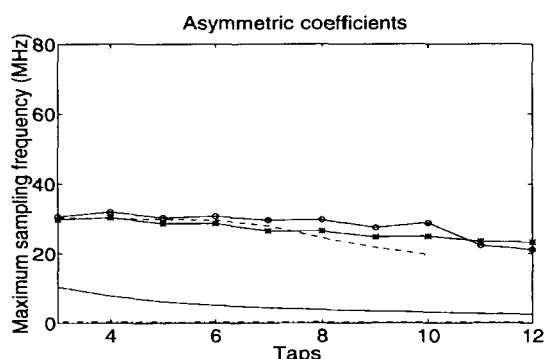


Fig. 14. Maximum sampling frequency as a function of asymmetric coefficients.

In Figures 15. and 16. the number of needed gates depending on the accuracy of samples and coefficients are shown. The number of coefficients used is 5. In the figures 17. and 18., the maximum sampling frequencies are shown for the same filters with same parameters and coefficients.

It must be emphasized that the maximum sampling frequency and the gate count depends on the values of coefficients when fixed implementations are used. This can be seen clearly in Figure 13., where the maximum sampling frequency may *increase* when the number of coefficients increases. However in the general case the maximum sampling frequency of the fixed implementations decreases little when the number of coefficients increases.

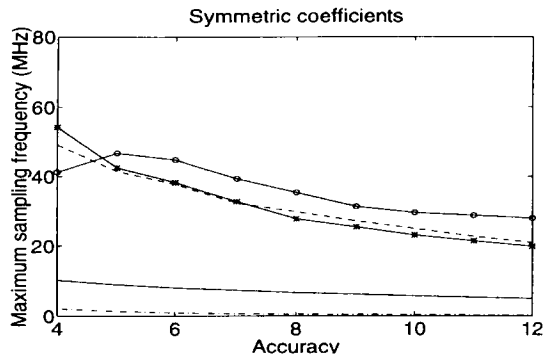


Fig. 17. Maximum sampling frequency as a function of the accuracy of samples and symmetric coefficients.

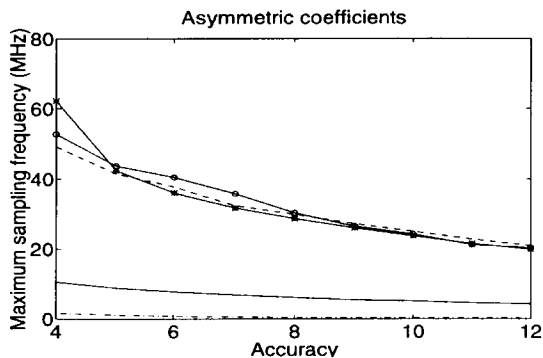


Fig. 18. Maximum sampling frequency as a function of the accuracy of samples and asymmetric coefficients.

7. Conclusion

The library developed is fully parameterized and generic, thus the synthesis results reported do not cover all cases, but rather give an impression of the possibilities the library can offer. However, the guidelines for choosing different kinds of filter architectures for different types of applications can be roughly derived from the results. With the parameters, the macro blocks can be tailored to the application requirements. Synthesizable VHDL has closed the gap between system design and hardware implementation. The library will be released commercially later but the final decision of the distributor has not been made yet.

Acknowledgments

This work has been a part of Finnish technology program ESV (Electronics design and manufacturing technologies) and is supported by the Technology Development Center (TEKES), Nokia Mobile Phones, Nokia Cellular Systems, Nokia Telecommunication/Transmission Systems, Nokia Data Communications, Nokia Research Center and Fincitec Ltd.

References

- [1] P. A. Findlay, B. Dickinson, M. Harris; "Production of Generic, Synthesizable ASIC Descriptions Using VHDL", VHDL-FORUM for Cad in Europe. Spring -93; pp. 41-52.
- [2] J. Isoaho, "DSP ASIC Development Based on Prototyping and Bit-modelling", Licentiate of Technology Thesis, Tampere University of Technology, Finland, 1992.
- [3] L. Ludeman, "Fundamentals of Digital Signal Processing", John Wiley & Sons, 1987.
- [4] J. Nousiainen, A. Nummela, J. Nurmi, H. Tenhunen; "Strategies for Developing and Modelling of VHDL Based Macrocell Library", The European Conference on Design Automation with The European Event in ASIC Design, EURO-ASIC-93; pp. 478 -482.
- [5] A. Oppenheim, W. Schaffer; "Discrete-Time Signal Processing", Prentice-Hall, 1989.
- [6] P. Pitkänen, J. Skyttä, T. Laakso; "Comparison of Digital Filter Architectures Using VHDL", EURO-VHDL'91, pp. 172 - 175.
- [7] J. Tieman, F. Yassa; "An IIR Filter Architecture with Programmable Structure, Order, and Coefficients", VLSI Signal Processing III, pp. 30-38. Ed. R. Brodersen, H. Moskovits, IEEE Press, 1988.
- [8] D. Werner, "The Cathedral II/III Module Library", March 1990, Leuven, Belgium.