DSP IN DEDICATED HARDWARE: RAISING VALUE ABSTRACTION FOR FIXED POINT IMPLEMENTATION

Wolfgang Pauli, Markus Pfaff, and Stefan Reichör

Markus Pfaff and Wolfgang Pauli are with the University of Applied Sciences
at Hagenberg/Austria, Hauptstr.117, A-4232 Hagenberg/Austria,
email: Wolfgang.Pauli@fh-hagenberg.at and pfaff@fh-hagenberg.at respectively, phone: +43 7236 3888 2420
web: www.fhs-hagenberg.ac.at
Stefan Reichör is with the RIIC – Research Institute for Integrated Circuits,
Freistädter Str. 315, Linz/Austria,
email: reichoer@riic.at, phone: ++43 732 2468 7117
web: www.riic.at

ABSTRACT

This abstract presents an arithmetic package for digital signal processing purposes that may be used with synthesizable VHDL code. This package provides means to raise the level of abstraction of a VHDL description without sacrificing the qualitiy of synthesis results. No changes to the hardware description language itself are needed to achive the desired effects, resulting in compatibility with existing simulation and synthesis tools.

Higher levels of abstraction enable a designer of dedicated hardware for digital signal processing to work more efficient and make fewer mistakes at the same time.

1. INTRODUCTION

Progress in the field of software engineering mainly originates in methods that raise the level of abstraction on which a designer has to express ideas to the compiler of the used programming language. The same applies to the area of digital hardware design where hardware description languages are in common use.

Different classification approaches for the concept of abstraction exist in the field of hardware design [1, 2]. The so called design cube [1] as shown in Fig. 1 classifies abstraction from a digital system designer's view. The package described in this paper raises the abstraction for fractional numbers. Such numbers are commonplace in the implementation of fixed point digital signal processing systems.

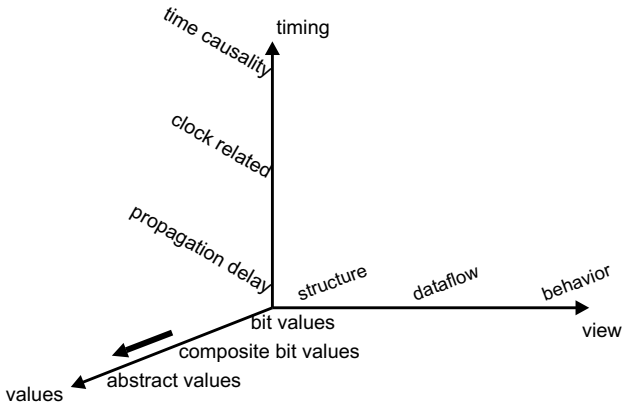

Figure 1: The so called design cube. The enlarged arrow in the lower left section shows the increase in value abstraction.

The proposed solution increases specifically value abstraction as shown by the enlarged arrow in Fig. 1.

The following section will give a short revision on the concept of fractional numbers. The two sections following are dedicated to the comparison of two VHDL descriptions, one using the `signed` data type while the other uses the much improved `fractional` data type.

2. FRACTIONAL NUMBER FORMAT

The number format typically used in fixed point digital signal processing systems is the L_2 or fractional number format [3, 4]. This format defines numbers in the range $[-1,1[$ with a fixed resolution of digits as follows:

$$x = a_0 \bullet a_1 \ldots a_{B-1} = -a^0 2^0 + \sum_{i=1}^{B-1} a_i 2^{-i} \qquad a_i \in \{0,1\} \quad (1)$$

The point defines the position of the comma in the number.

When we talk about the fractional number format we often say that we use a 1.X format, where X presents the amount of binary digits following the comma. Digital signal processors as they exist as common off the shelf products make use of this number format in their fixed point arithmetic units.

3. VHDL IMPLEMENTATION

Unfortunately the fractional number format is not available as a standard format to VHDL (or Verilog) developers. This is one of the reasons making the implementation of a fixed point digital signal processing system on dedicated hardware (e.g. ASIC, FPGA) a difficult task. Only binary integer numbers are predefined as data types for synthesis purposes.

3.1 Standard Data Type `signed`

The data type `signed` is defined in the standardized VHDL package `numeric_std` [5]. Alternatively the package `std_logic_arith` originally introduced by the synthesis tool vendor Synopsys may be used, because it makes similar definitions.

While this data type is very useful for integer like arithmetic it doesn't provide a good basis for digital signal processing. While a larger bit width of a `signed` value will in-

crease the *value range* of `signed` number, it has the meaning of *increased precision* for fractional numbers.

Examples of VHDL descriptions for digital signal processing [6, 2] make use of this data type, but thereby deviate from the standard L_2 number format commonly used. The resulting code tends to hide the architecture used for signal processing. In addition the signal range is highly inconsistent, because the bitwidth has to be adapted after arithmetic operations.

An example may illustrate the complicated form such a VHDL description will involuntarily take. The choosen code stems from the implementation of an IIR filter block of second order. Such a block is depicted in Fig. 2.

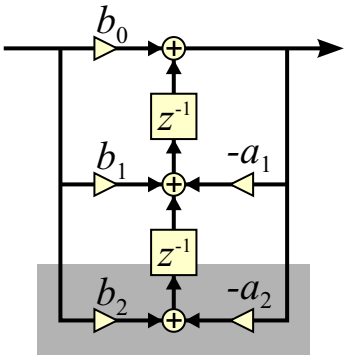

Figure 2: An IIR filter block of 2nd order. The shaded area emphasizes on the part which serves as an example for VHDL implementation.

We focus on the shaded area in Fig. 2 for easier comparison. This area contains one addition and one multiplication of the signal with a constant filter coefficient. This operation computes the result for a register used to delay for one sampling period. A first code example is confined to the use of the predefined data type `signed`:

```
 1 IIR2ndOrder : process(iClk, iResetAsync) is
 2   variable Temp32 : signed(31 downto 0);
 3   variable Temp16 : signed(15 downto 0);
 4 begin
 5   if iResetAsync = '1' then
 6     ...
 7   elsif rising_edge(iClk) then
 8     ...
 9     Temp32 := Shift_Left(iDdry * "11100001", 1) +
10               Shift_Left(oDWet * "10111100", 1);
11     Temp16 := Temp32(31 downto 16);
12
13     if Temp16(15) = '1' then
14       Temp16 := Temp16+1;
15     end if;
17     Delay2ndOrder <= Temp16;
18       ...
19   end if;
20 end process IIR2ndOrder;
```

This code is cluttered by temporary assignments. The clear structure shown in Fig. 2 is concealed and the constant values representing the filter coefficients (lines 9 and 10) are represented in a format that essentially demands explanation to be understandable. This format may vary throughout a complete system.

In this example only two operations are shown, but for more complex structures the resulting code will simply be unreadable. The main problem here is the multiplication of 1.X values. The result must be truncated and the position of the comma must be corrected, because the result should again be in the 1.X format. Simple truncation of values may lead to limit-cycles which should be avoided, because limit-cycle oscillation will be noticable at the output of a signal processing system long after the input excitation has faded out for an IIR filter. The code needed for this step (shown in lines 13 to 15) brings in additional complexity.

As an alternative the number format may be changed from operation to operation, but this will inevitably lead to an even worse situation in terms of code maintainablility.

3.2 The Package `Fractional`

The package `Fractional` provides the data type `FRACT` and defines operations on this type that are needed for typical signal processing applications. The data type `FRACT` is defined to be as close to the mathematical definition in equation 1 as possible.

The index range of this type starts at 0 and runs down the negative axis. No digit shifting is needed to read out the right number from a binary representation.

The following code implements the very same funcionality as the code above, but makes use of the package `Fractional`:

```
 1 IIR2ndOrder : process(iClk, iResetAsync) is
 2 begin
 3   if iResetAsync = '1' then
 4   ...
 5   elsif rising_edge(iClk) then
 6     Delay2ndOrder <=
 7       TruncAbsVal(iDdry * -0.784562) +
 8       TruncAbsVal(oDWet * -0.876452);
 9     ...
10   end if;
11 end process IIR2ndOrder;
```

The readablity is improved a lot, because the code directly reflects the structure given in the shaded area of Fig. 2. In this example we have two multiplications. The sample `iDdry` which presents the input of the IIR filter has a bit width of e.g. 16 bit and the result of the multiplication also leads to a bit width which the input sample determines. This means that in the operational part of the package the bit width of the `FRACT_REAL` value is adapted to `iDdry`. In general the multiplication can be processed with two different bit widths of the arguments but the result is always in the width of the higher precision.

As mentioned before, the truncation which is performed by the multiplication can lead to limit-cycles and must be avoided. The package provides the function `TruncAbsVal` for this purpose. After the usage of this function no cycling is possible.

The package allows the designer to use different bitwidths for torough optimization in a digital signal processing system. Therefore a `resize` function is implemented which takes as parameters the input vector and the desired bit width to resize to.

Filter design tools like SPTOOL, a part of the Matlab signal processing toolbox, deliver the values for the L_2 scaled filter coefficients in real format. As mentioned before these values spin the range of $[-1, 1[$. To express such constants in a readable format the data type `FRACT_REAL` is defined as a subtype of the VHDL predefined type `real`. In the example

the coefficients are given as constant values (expressed as literals for the sake of simplicity) of type FRACT_REAL.

Although the type real is generally *not* synthesizable using the synthesis tools on the market, the multiplication with a constant of this type is synthesizable in this special case. This is accomplished with a type conversion from FRACT_REAL to FRACT which is hidden in the operational part of the package [7].

No laborious conversions like in the example of section 3.1 have to be done, although conversions functions from and to the FRACT_REAL and FRACT data types are supplied by the package if needed.

4. SYNTHESIS RESULTS

Raising abstraction will in general detoriate synthesis results. This is not the case when value abstraction as proposed by this paper is used. To proof this statement different synthesis tools from three vendors were used. The point is that the usage of packages like the one described here can only be recommended, if the provided functions do not deflect the hardware result in respect of area and delay which are the important factors for cost of dedicated hardware. For complex filter structures like Fig. 3 this would lead to an overhead in terms of resources. The mentioned facts make it evident that such an abstraction must be done very carefully.

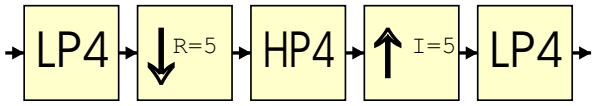

Figure 3: A complex bandpass filter structure. Abstraction which detoriates the synthesis result leads to an overhead in terms of resources. This filter was implemented for audio signals using a Xilinx FPGA of the Spartan 2 family.

The base for the synthesis result comparison is the 2nd Order Direct Form shown in Fig. 2. Digital signal processing systems and especially complex filters (e.g. Fig. 3) of higher order are often implemented in cascade form. This means that for a realization of a 4th order filter we use two 2nd order IIR structures in serial connection. In the synthesis we compare only the IIR structure printed in Fig. 2 because we are in relation close to the whole system in terms of area and delay. To emphasize the shortness of the code and to give an example how such structure is implemented, the full VHDL sample is printed.

```
use work.Fractional.all;

entity DspIirOrd2 is
  port (
    -- sequential elements in architecture
    iClk        : in  std_ulogic;
    iResetAsync : in  std_ulogic;
    -- input signals of 2nd order IIR
    iDdry       : in  fract(0 downto -15);
    iValDry     : in  std_ulogic;
    -- output signals of 2nd order IIR
    oDwet       : out fract(0 downto -15);
    oValWet     : out std_ulogic);
end DspIirOrd2;

architecture RTL of DspIirOrd2 is
  signal DWet, DWetUnreg : fract(0 downto -15);
  signal Delay1stOrder   : fract(0 downto -15);
  signal Delay2ndOrder   : fract(0 downto -15);
begin

 IIR2ndOrder : process(iClk, iResetAsync) is
 begin
  if iResetAsync = '1' then
    DWet          <= (others => '0');
    Delay1stOrder <= (others => '0');
    Delay2ndOrder <= (others => '0');
    oValWet       <= '0';
  elsif rising_edge(iClk) then
   if iValDry = '1' then
     DWet          <= DWetUnreg;
     oValWet       <= '1';
     Delay1stOrder <=
               TruncAbsVal(iDdry * (-0.161146))  +
               TruncAbsVal(DWetUnreg * 0.893874) +
               TruncAbsVal(DWetUnreg * 0.893874) +
               Delay2ndOrder;
     Delay2ndOrder <=
               TruncAbsVal(iDdry * 0.101068) +
               TruncAbsVal(DWetUnreg * 0.845648);
   else
     oValWet <= '0';
   end if;
  end if;
 end process IIR2ndOrder;

 DWetUnreg <= Delay1stOrder +
              TruncAbsVal(iDdry * 0.101068);
 oDwet     <= DWet;

end architecture RTL;
```

4.1 Leonardo Spectrum Version 2003b

This test was done with Leonardo Spectrum Version 2003b. The resource report which was created by the tool is directly printed below.

4.1.1 *Synthesis results using* signed

```
-- Start optimization for design
.work.DspIirOrd2.RTL

Using wire table: xis2150-6_avg

Pass  Area  Delay  DFFs  PIs  POs   --CPU--
     (LUTs)  (ns)                  min:sec
 1     542     29    49   19   17     00:02
 2     542     29    49   19   17     00:02
 3     542     29    49   19   17     00:02
 4     542     29    49   19   17     00:02
```

4.1.2 *Synthesis results using* fractional

```
-- Start optimization for design
.work.DspIirOrd2.RTL

Using wire table: xis2150-6_avg

Pass  Area  Delay  DFFs  PIs  POs   --CPU--
     (LUTs)  (ns)                  min:sec
 1     542     29    49   19   17     00:02
 2     542     29    49   19   17     00:02
 3     542     29    49   19   17     00:02
 4     542     29    49   19   17     00:02
```

The two results clarify, that both code solutions deliver the same area and delay for dedicated hardware. The amount of data flip-flops is determined by the bit width of the delay structure and the registered output. To inform a following block that data is delivered a synchronization signal (oValWet) is present.

4.2 Synopsys Version 2003.09

4.2.1 Synthesis results using `signed`

```
****************************************
Report : area
Design : DspIirOrd2UsingSigned
Version: 2001.08-SP2
Date   : Mon Apr 19 17:40:01 2004
****************************************

Library(s) Used:

csx_HRDLIB (File: /eda/rev1/share/libraries/
    ams_dir/synopsys/csx_3.3V/csx_HRDLIB.db)

Number of ports:                36
Number of nets:                389
Number of cells:               147
Number of references:           22

Combinational area:       308253.406250
Noncombinational area:    30048.199219
Net Interconnect area:    46386.000000

Total cell area:          338301.605469
Total area:               384687.593750
```

4.2.2 Synthesis results using `fractional`

```
****************************************
Report : area
Design : DspIirOrd2UsingFractional
Version: 2001.08-SP2
Date   : Mon Apr 19 20:23:41 2004
****************************************

Library(s) Used:

csx_HRDLIB (File: /eda/rev1/share/libraries/
    ams_dir/synopsys/csx_3.3V/csx_HRDLIB.db)

Number of ports:                36
Number of nets:                389
Number of cells:               147
Number of references:           22

Combinational area:       308253.406250
Noncombinational area:    30048.199219
Net Interconnect area:    46386.000000

Total cell area:          338301.605469
Total area:               384687.593750
```

These values again confirm that there is no synthesis result detoriation because of the done abstraction. One point which must be mentioned is that `Synopsys` does not support the `FRACT_REAL` data type for static synthesis purposes. Some slight modifications must be done to get to the synthesis result.

4.3 SynplifyPro Version 5.7.1

4.3.1 Synthesis results using `signed`

Pass	CPU time	Worst Slack	Luts / Registers
1	0h:0m:2s	-6.88ns	581 / 49
2	0h:0m:2s	-6.88ns	581 / 49
3	0h:0m:2s	-6.88ns	581 / 49

4.3.2 Synthesis results using `fractional`

Pass	CPU time	Worst Slack	Luts / Registers
1	0h:0m:2s	-2.99ns	483 / 49
2	0h:0m:2s	-2.99ns	483 / 49
3	0h:0m:2s	-2.99ns	483 / 49

This printed summarys show that both solutions deliver nearly the same results. In this case the implementation using the `fractional` type result in a smaller usage of `combinatorical logic` in comparison to the `signed` type. The slack value which determines the backup from the calculated maximum possible clock frequency to the constrained clock frequency differs also. The `signed` version is slightly faster then the `fractional` one but in terms of area the `fractional` version has advantages. As summary we can say that there is also no significant detoriation of the synthesis result.

5. CONCLUSION

The usage of a simple package makes implementation of fixed point digital signal processing algorithms in dedicated hardware a lot easier. Code readability is improved and the designer is free concentrate on the structure of the algorithm rather than artificially complicated bit string arithmetic.

To get even closer to a architecture centric approach further functionality can be added to the package. One such extension is the addition of saturation logic which prevents the two's complement overflow problem.

REFERENCES

[1] Wolfgang Ecker and Michael Hofmeister, "The design cube: A model for vhdl designflow representation," in *Proceedings of the Conference on European Design Automation.*, Hamburg, Germany, 1992, pp. 752–757.

[2] Klaus ten Hagen, *Abstrakte Modellierung digitaler Schaltungen*. Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[3] Israel Koren, *Computer Arithmetic Algorithms*. AK Peters, 2nd edition, 2003.

[4] Michael J. Flynn and Stuart F. Oberman, *Advanced Computer Arithmetic Design*. John Wiley & Sons, Inc., New York 2001.

[5] Markus Pfaff and Markus Schutti, "Das VHDL-Package ieee.numeric_std: Einsatzbereich, Grenzen, Risiken," in *Surface mount technologie, electronic systems & solutions, technologie, circuits & tools, hybrid and advanced packaging technologies: Tagungsband SMT ES&S Hybrid, Internationale Messe und Kongress für Systemintegration, 22.-24. Mai 1997.*, Berlin, Offenbach, April 1997, pp. 661–670, VDE-Verlag.

[6] Peter J. Ashenden, *The Designer's Guide to VHDL*. Morgan Kaufman Publishers, 2 edition, 2001

[7] IEEE Std 1076-2002 *IEEE Standard VHDL Language Reference Manual*