

# Design of fixed-point rounding operators for VHDL-2008

Nikolaos Kavvadias and K. Masselos  
{nkavv, kmas}@uop.gr

Department of Computer Science and Technology,  
University of Peloponnese,  
Tripoli, Greece

\* Special thanks to C. Lezos for presenting this paper at the DASIP 2012 venue

23 October 2012



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ  
UNIVERSITY OF PELOPONNESE



# Introduction and motivation

- The design of sophisticated DSP platforms involves specifications at an increasingly raised abstraction level to confront with the growing complexity-productivity gap
- MATLAB and other high-level programming environments enable rapid algorithm development and prototyping
- An interesting aspect of such frameworks is low-level design facet generation
  - ANSI C reference fixed-/floating-point model generation
  - Synthesizable HDL code
- Fixed-point rounding (FXPR) is essential for the implementation of fixed-point datapaths
- ☞ We introduce generic and parameterized VHDL descriptions of fixed-point rounding operators

# Fixed-point rounding in contemporary EDA tools

- Rounding operators control the numerical precision involved in a series of computations
- The FPT (Fixed-Point Toolbox) MATLAB plugin defines fixed-point data types
- Besides `floor` (round towards  $-\infty$ ) and `ceil` ( $+\infty$ ), it also defines `fix` (zero), `round` (nearest; ties to  $+\infty$ ), `nearest` (nearest; ties to greatest absolute value) `convergent` (nearest; ties to closest even)
- Supported by the MATLAB HDL Coder
- GraphLab, AccelDSP (discontinued) and Synopsys Symphony Model Compiler appear to provide certain support to FXPR
- VHDL-2008 adds fixed-point data types (`ufixed`, `sfixed`) and primitives for arithmetic, scaling and operand resizing
- We propose generic designs that achieve better timing by about 30% and similar area demands to the HDL Coder

# VHDL-2008 native types for fixed-point arithmetic

- Fixed-point arithmetic is seen as an integral representation variant where a binary point is defined as a notational artifact to signify negative integer powers
- VHDL-2008 supports signed (`sfixed`) and unsigned (`ufixed`) fixed-point arithmetic
- `sfixed`: Assuming integer part width  $IW > 0$ , and a fractional part of  $-FW < 0$ , the representable data range is  $2^{IW-1} - 2^{|FW|}$  to  $-2^{IW-1}$
- `ufixed`:  $2^{IW} - 2^{|FW|}$  to  $-2^{IW-1}$
- Resolution:  $2^{|FW|}$
- VHDL examples

```
signal fxp1 : sfixed(4 downto -5);
signal fxp2 : ufixed(7 downto -8);
-- using generics
signal fxp3 : sfixed(IW-1 downto -FW);
```

# Basic resizing and rounding

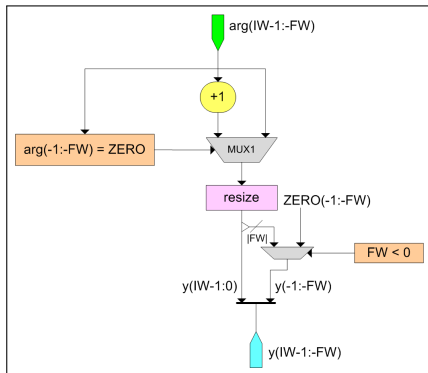
- `resize` is a VHDL library function that can be used as a rounding and saturation primitive for adjusting the size of fixed-point operands
- Input: a fixed-point operand (*arg*), a left and a right index bound and two additional arguments that specify the rounding (underflow) and saturation (overflow) mechanisms
- Output: resized operand (*y*)
- Rounding of quantized integers, defined by  $IW > 0$ ,  
 $-FW > 0$

```
if (arg'low > 0) then
  y := resize(arg, arg'high, arg'low);
  return y;
end if;
```

- Rounding of quantized fractionals, defined by  $IW < 0$ ,  
 $-FW < 0$ , is trivial (assignment to zero)

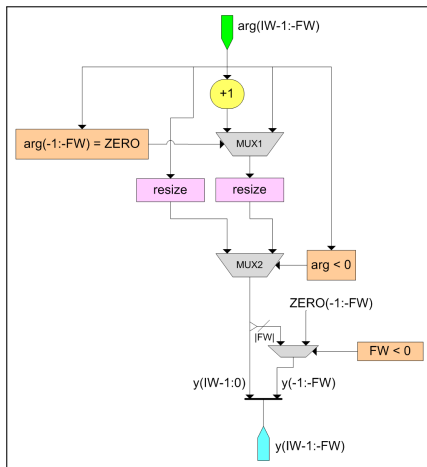
# Implementation of the ceil operator

- $arg$  is increased by 1 if it has a non-zero fractional part, otherwise it is passed directly to `resize`
- The 2nd mux allows for fixed-point numbers with any  $FW$  value
- For hardware synthesis, it is always eliminated at design elaboration time



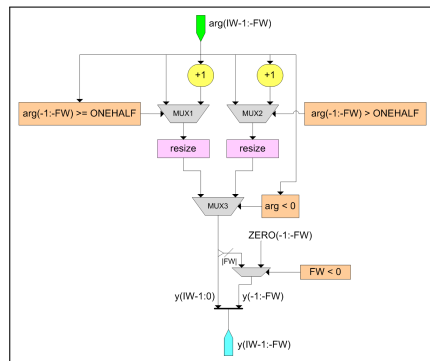
# fix and floor

- If  $arg < 0$ , then it is either directly resized or first incremented by 1, depending on the comparison of its fractional part to zero
- Unsigned fix (fixu) only involves resizing  $arg$
- floor is implemented by fixu



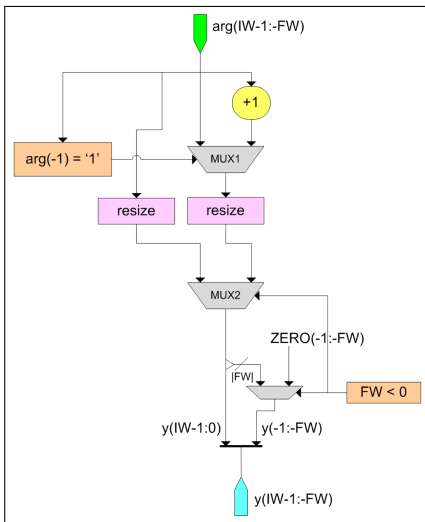
# round

- Requires comparisons to the value of  $\frac{1}{2}$  (ONEHALF)
- If the fractional part of a positive  $arg$  is  $\geq \frac{1}{2}$ , then it has to be incremented prior resizing
- The complementary rule applies for a negative  $arg$
- `roundu` requires a subset of this circuit



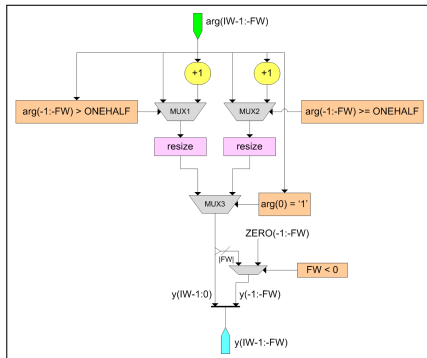
# nearest

- If the fractional part of  $arg$  is  $\geq \frac{1}{2}$ , then its incremented value needs to be resized
- This is performed by the  $arg(-1) = '1'$  comparator
- Additional multiplexing can be eliminated at design elaboration time
- `nearestu` shares the same circuit



# convergent

- Requires comparisons to the value of  $\frac{1}{2}$  as well as to determine whether the integral part of  $arg$  is odd
- The latter is needed for resolving a tie
- The integral part is selected from multiplexers MUX1 and MUX2, based on the results of the corresponding comparators

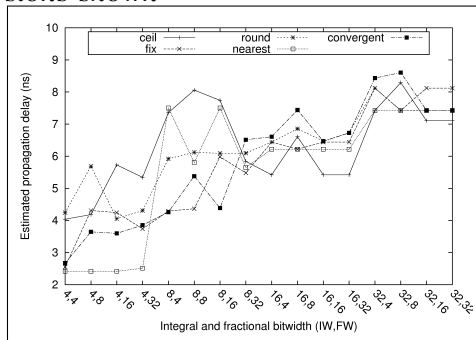


# Complete VHDL source code for the convergent operator

```
function convergent (arg : sfixed) return sfixed is
  variable result: sfixed(arg'high downto arg'low);
  variable onehalf: std_logic_vector(-arg'low-1 downto 0) := (others => '0');
begin
  if (arg'high <= 0) then
    result := (others => '0'); return result;
  end if;
  if (arg'low > 0) then
    result := resize(arg, arg'high, arg'low); return result;
  end if;
  onehalf(-arg'low-1) := '1';
  if (arg(0) = '1') then
    if (to_slv(arg(-1 downto arg'low)) >= onehalf) then
      result := resize(arg + 1, arg'high, arg'low);
    else
      result := resize(arg, arg'high, arg'low);
    end if;
  else
    if (to_slv(arg(-1 downto arg'low)) > onehalf) then
      result := resize(arg + 1, arg'high, arg'low);
    else
      result := resize(arg, arg'high, arg'low);
    end if;
  end if;
  if (arg'low < 0) then
    result(-1 downto arg'low) := (others => '0');
  end if;
  return result;
end function convergent;
```

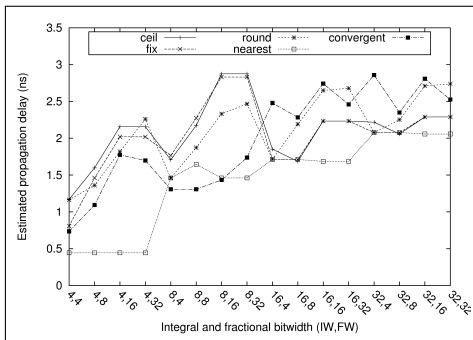
# Speed measurements (XC3S200 Spartan-3 FPGA)

## ■ sfixed versions shown



- Estimated propagation delay is less than 9ns for the largest configuration
- unfixed circuits are faster to sfixed ones by 17.8%
- (8, 32) operators are faster than their (8, 16) counterparts due to using the MUXF5 wide multiplexer primitive

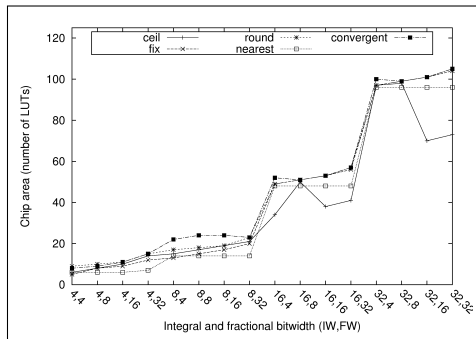
# Speed measurements (XC6VLX75T Virtex-6 FPGA)



- Estimated propagation delay is less than 3ns for the largest configuration
- Designs are faster by 67% compared to XC3S200
- nearests has the fastest implementation

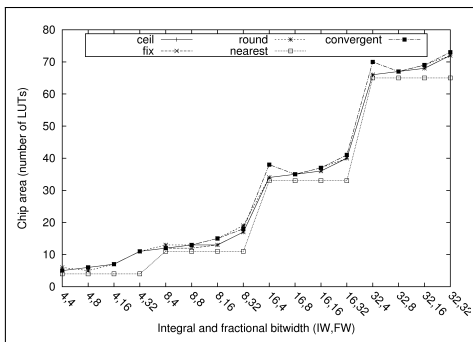
# Chip area measurements (XC3S200 Spartan-3 FPGA)

- sfixed versions are shown



- Requirements range from a few to 105 LUTs
- 27% less LUTs required for unfixed operators
- ceil is the smallest operator

# Chip area measurements (XC6VLX75T Virtex-6 FPGA)



- Up to 74 LUTs required for any operator
- 6-input LUTs appear more densely populated
- nearest is the smallest operator; round for ufixed arithmetic

# Architectural comparison to architectures generated by the Simulink HDL coder

- Performed comparisons to fixed-point rounding VHDL implementations generated by the MATLAB 7.8.0 (R2009a) Simulink HDL coder (*hdlcoder*)
- For a fair comparison, the *proposed* units were rewritten for the `numeric_std` package

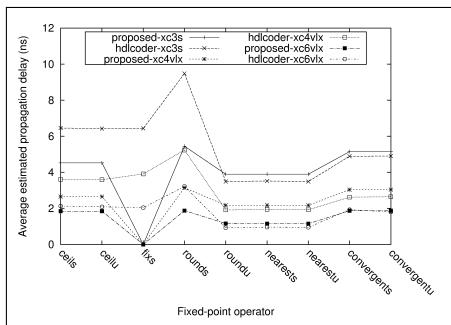
## ■ *proposed*

- Generic, portable descriptions
- The same principles applicable to floating-point formats
- No decrement operations needed
- No extended internal precision

## ■ *hdlcoder*

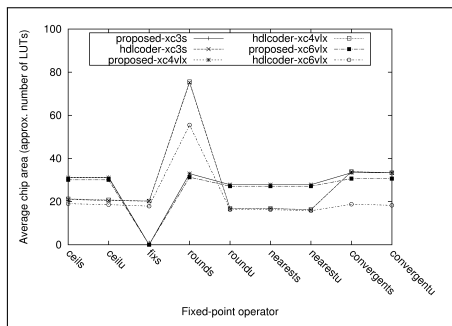
- Non-portable descriptions using bitwise tricks
- Not applicable to floating-point formats
- Decrement operations
- Need for an additional bit of accuracy

# Speed measurements comparison to the Simulink HDL Coder



- The *proposed* *s*fixed units outperform the *hdlcoder* ones, by 31.3%, 27.7% and 27.6%, respectively
- Except *ceilu*, unsigned *hdlcoder* units are faster
- ☞ This is due to unsigned bitwise tricks applied by *hdlcoder* (e.g. *convergentu* uses a reduction OR gate)

# Chip area measurements comparison to the Simulink HDL Coder












- The *proposed* *s*fixed units are comparable to the *hdlcoder* ones
- The *proposed* *r*ounds requires much less than its counterpart
- However, *hdlcoder* units consume much less area (38.5%-65.7%)

# Conclusions

- Novel schemes for implementing fixed-point binary rounding have been introduced
- Implemented using vendor-independent, generic, portable, standard-compatible VHDL
- Thorough experimental measurements have been shown over three representative FPGA devices
- Comparisons to rounding units generated by the Simulink HDL coder (*hdlcoder*) reveal that the proposed designs for signed arithmetic are faster by 30% with comparable area demands
- *hdlcoder* designs are generally faster for unsigned arithmetic
- The fixed-point extensions library is available as open-source: [http://www.opencores.org/project,fixed\\_extensions](http://www.opencores.org/project,fixed_extensions)

# References

-  S. Roy and P. Banerjee, “An algorithm for trading off quantization error with hardware resources for MATLAB-based FPGA design,” *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 886–896, July 2005.
-  MATLAB Fixed-Point Toolbox. [Online]. Available: <http://www.mathworks.com/products/fixed/>
-  R. Yates, “Fixed-point arithmetic: An introduction,” Digital Signal Labs, Technical reference, July 7 2009.
-  Algorithmic C data types. [Online]. Available: [http://www.mentor.com/products/esl/high\\_level\\_synthesis/ac\\_datatypes](http://www.mentor.com/products/esl/high_level_synthesis/ac_datatypes)
-  *IEEE 1076-2008 Standard VHDL Language Reference Manual*, Jan. 2009.
-  D. Bishop. VHDL-2008 support library. [Online]. Available: <http://www.eda.org/fphdl/>
-  Simulink HDL Coder. [Online]. Available: <http://www.mathworks.com/products/slhdlcoder/>
-  B. L. Gal and E. Casseau, “Word-length aware DSP hardware design flow based on high-level synthesis,” *Integration, the VLSI Journal, Elsevier*, vol. 62, pp. 341–357, March 2011.
-  Xilinx home page. [Online]. Available: <http://www.xilinx.com>