

# 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 PELLOPONNESE

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

Design of fixed-point rounding operators for VHDL-2008

Design of fixed-point rounding operators for VHDL-2008

2012-09-29

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



- No additional comments

# 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

- 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

- Recent EDA tools provide algorithm specification and code generation facilities that enable a quick path for concept to implementation
  - The subject of fixed-point rounding is generally underestimated in its importance and difficulty
  - Most hardware compilation frameworks either lack the support of these operators or provide specialized and non-portable implementations
  - Our solution is compatible with both the latest (VHDL-2008) and earlier versions of the VHDL standard (up to at least VHDL-1993)
- 📄 A complete implementation is available at the Opencores site:  
[http://www.opencores.org/project,fixed\\_extensions](http://www.opencores.org/project,fixed_extensions)

# 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

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

Design of fixed-point rounding operators for VHDL-2008

Design of fixed-point rounding operators for VHDL-2008

2012-09-29

└ Fixed-point rounding in contemporary EDA tools

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

- There does not exist a language-independent standard for fixed-point arithmetic
- GraphLab is a high-level synthesis (HLS) tool supporting a subset of MATLAB. However, it is unreleased
- AccelDSP is a discontinued Xilinx [9] product promoted as a designer assist for DSP algorithm compilation to FPGA-oriented hardware
- Synopsys Symphony HLS includes a synthesizable fixed-point high-level IP model library. Unfortunately, we were not able to access either a demo or sufficient documentation/examples
- The Simulink HDL coder utilizes an undisclosed vendor-dependent library for supporting fixed-point arithmetic
- The proposed designs are highly-optimized combinatorial functions so that they can be included as atomic operations in modern soft-core microprocessors

# 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);
```

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

2012-09-29

### VHDL-2008 native types for fixed-point arithmetic

#### 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);
signal fxp3 : sfixed(IW-1 downto -FW);
```

- Fixed-point arithmetic provides an inexpensive means for improved numerical dynamic range, when quantization and overflow effects can be tolerated
- Actually, the VHDL fixed-point standard package allows for greater freedom (e.g.  $FW < 0$ ) but in this work the following are assumed:
  - The binary point is located between the  $2^0$  and  $2^{-1}$  powers
  - At least one fractional bit is defined (i.e. the fixed-point numbers are not degenerate integers)
  - At least one integral bit is defined

# 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;
```

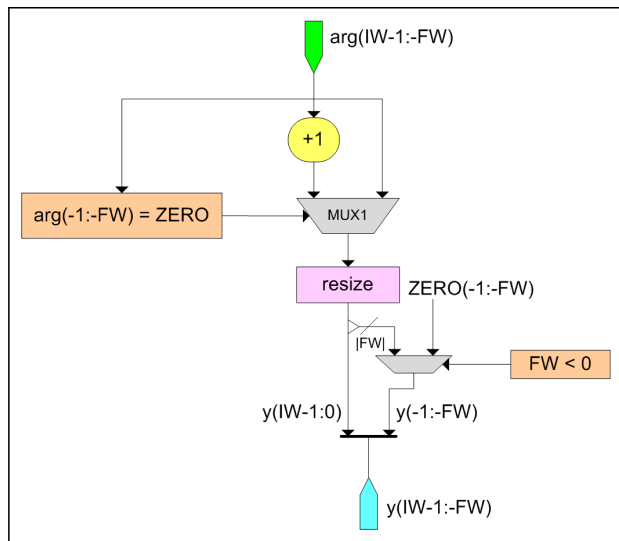
- Rounding of quantized fractionals, defined by  $IW < 0$ ,  $-FW < 0$ , is trivial (assignment to zero)

- `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)

- While different underflow (rounding to nearest and truncation) and overflow (saturation or wrap-around) schemes are supported by `resize`, they do not provide the extensive functionality required for emulating MATLAB rounding
- All the proposed operators use `resize` with the default overflow/underflow modes (`fixed_round`, `fixed_saturate`) as an essential building block

# 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



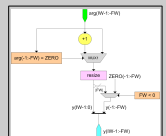
## Design of fixed-point rounding operators for VHDL-2008

2012-09-29

### Implementation of the ceil operator

#### Implementation of the ceil operator

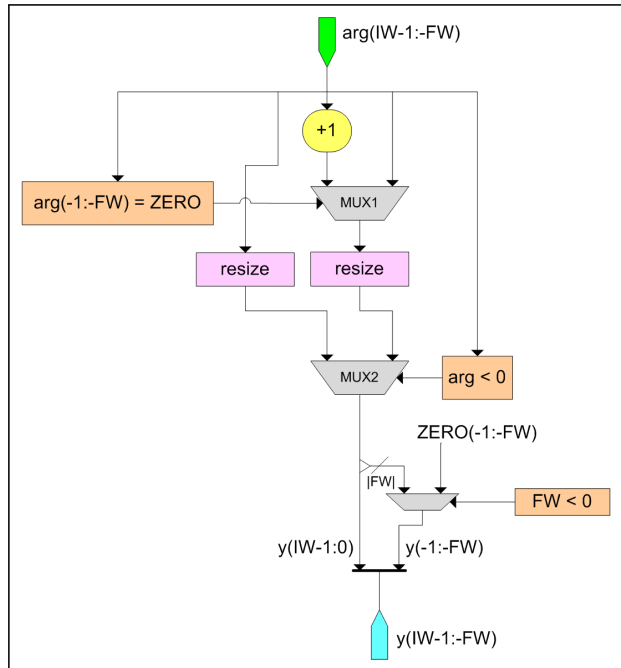
- $arg$  is increased by 1 if it has a non-zero fractional part, otherwise it is passed directly to `resize`
- The 2nd mux allows for fixed-point numbers with any  $FW$  value
- For hardware synthesis, it is always eliminated at design elaboration time



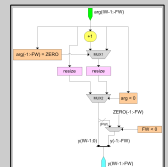
- Same hardware for unsigned and signed arithmetic
- Vector upper and lower bounds ( $arg'HIGH$ ,  $arg'LOW$ ) are known at design compile-time; thus this structure can be eliminated
- This runtime check can be omitted if it is guaranteed that  $arg$  is not a quantized integer

# 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**



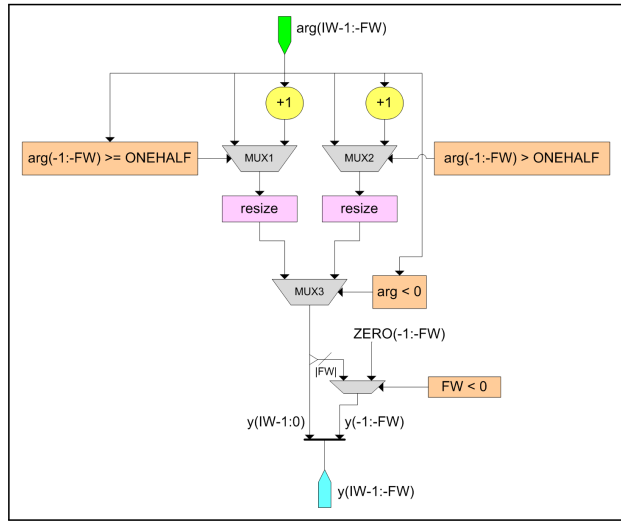
- If  $arg < 0$ , then it is either directly resized or first incremented by 1, depending on the comparison of its fractional part to zero
- Unsigned **fix** (**fixu**) only involves resizing  $arg$
- **floor** is implemented by **fixu**



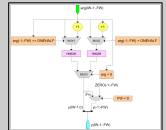
- **floor** rounds towards  $-\infty$  which means that it implements a consistent scheme for either positive or negative values (signed arithmetic) or if considering only magnitude (unsigned)

# 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



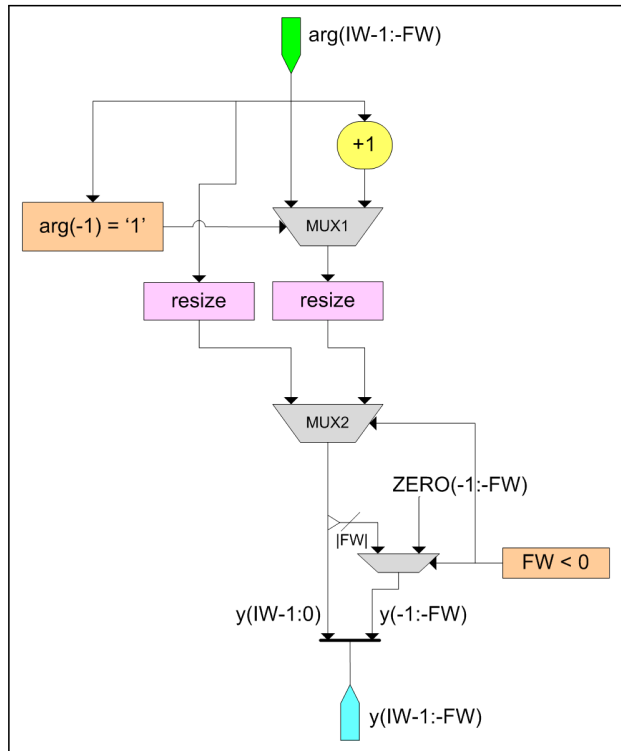
- Requires comparisons to the value of  $\frac{1}{2}$  (ONEHALF)
- If the fractional part of a positive *arg* is  $\geq \frac{1}{2}$ , then it has to be incremented prior resizing
- The complementary rule applies for a negative *arg*
- `roundu` requires a subset of this circuit



- No additional comments

# 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



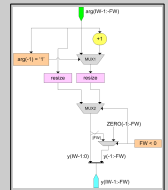
## Design of fixed-point rounding operators for VHDL-2008

2012-09-29

nearest

nearest

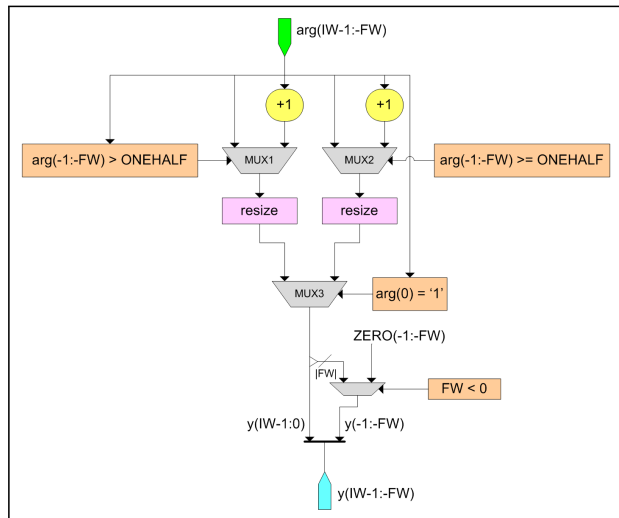
- If the fractional part of  $arg$  is  $\geq \frac{1}{2}$ , then its incremented value needs to be resized
- This is performed by the  $arg(-1) = '1'$  comparator
- Additional multiplexing can be eliminated at design elaboration time
- nearestu shares the same circuit



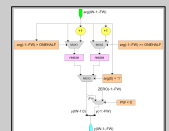
- No additional comments

# 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



- 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



- The comparator on the left checks whether the fractional part of  $arg$  is  $> ONEHALF$
- The comparator on the right performs  $\geq ONEHALF$

# 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;
```

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

2012-09-29

— Complete VHDL source code for the convergent operator

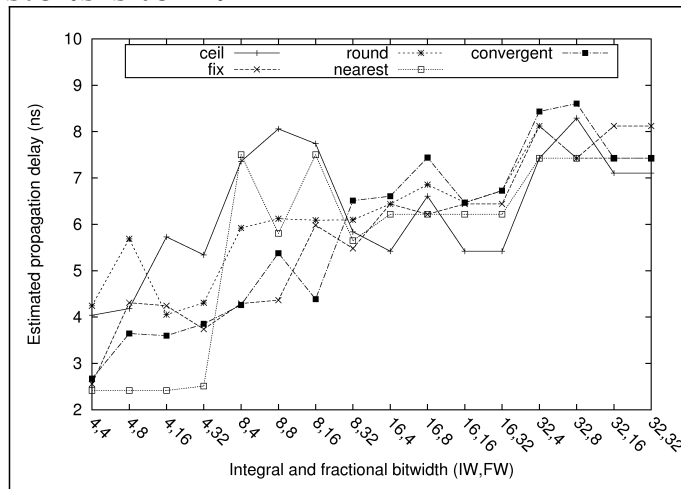
Complete VHDL source code for the convergent operator

```
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;
```

- This version can be even more generalized by avoiding constant indexing of the *arg* vector
- By providing *IW*, *FW* as input variables (or VHDL-2008 package generics), the LSB (Least Significant Bit) of the integral part (indexed by 0) can be referred to by *arg'HIGH-IW*
- Correspondingly, the MSB index (-1) of the fractional part is equivalent to *arg'low+FW-1*

# 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

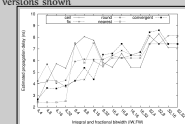
## Design of fixed-point rounding operators for VHDL-2008

2012-09-29

### Speed measurements (XC3S200 Spartan-3 FPGA)

#### Speed measurements (XC3S200 Spartan-3 FPGA)

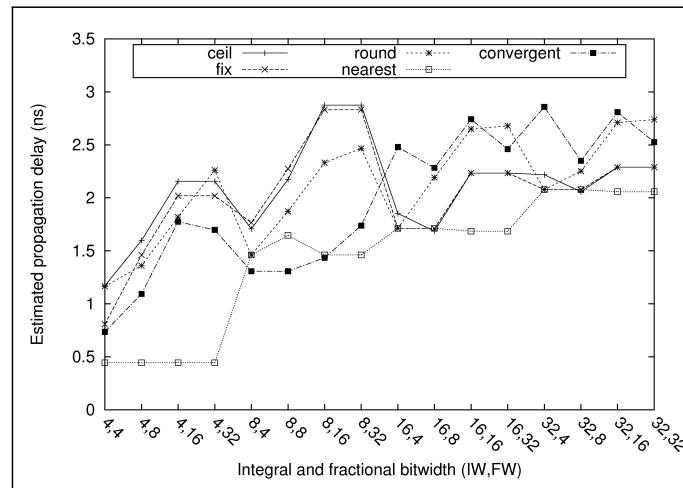
##### ■ sfixed versions shown



- 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

- We have obtained speed and chip area measurements for different values of the  $IW, FW$  parameters
- For each point in the parameter set and for both unfixed and sfixed variants of the designs, the timing (minimum propagation delay) and area requirements are measured for three representative FPGA processes: the 90nm Spartan-3 and Virtex-4 5-input LUT and the 40nm Virtex-6 6-input LUT process
- In these slides, only results for the two extremes, XC3S200 and XC6VLX75T are shown
- The logic synthesis tool used is Xilinx Webpack ISE 12.3i
- fixu, fixs and flooru are not shown, since they are optimized out to plain wiring by logic synthesis

# Speed measurements (XC6VLX75T Virtex-6 FPGA)



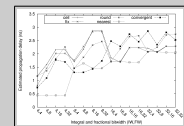
- Estimated propagation delay is less than 3ns for the largest configuration
- Designs are faster by 67% compared to XC3S200
- nearests has the fastest implementation

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

2012-09-29

└ Speed measurements (XC6VLX75T Virtex-6 FPGA)

### Speed measurements (XC6VLX75T Virtex-6 FPGA)

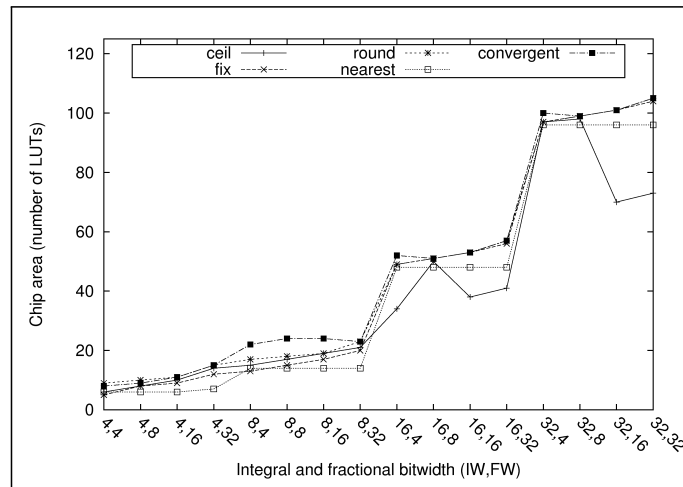


- Estimated propagation delay is less than 3ns for the largest configuration
- Designs are faster by 67% compared to XC3S200
- nearests has the fastest implementation

- No additional comments

# 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

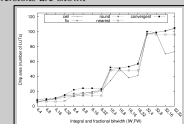
## Design of fixed-point rounding operators for VHDL-2008

2012-09-29

└ Chip area measurements (XC3S200 Spartan-3 FPGA)

### Chip area measurements (XC3S200 Spartan-3 FPGA)

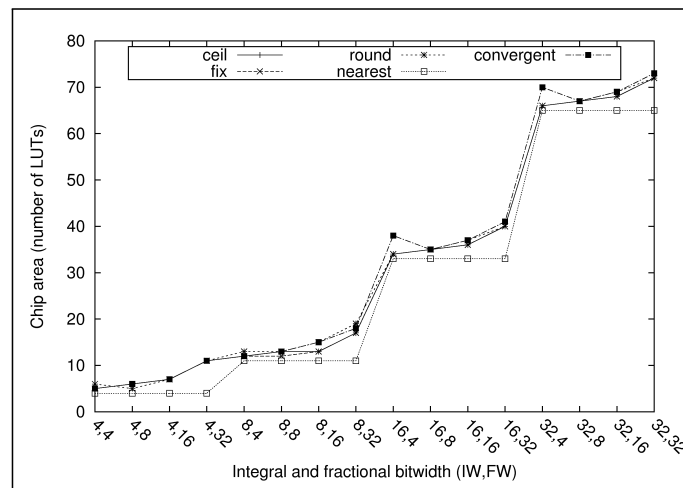
- sfixed versions are shown



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

- No additional comments

# Chip area measurements (XC6VLX75T Virtex-6 FPGA)



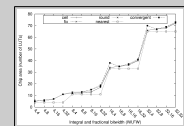
- Up to 74 LUTs required for any operator
- 6-input LUTs appear more densely populated
- nearest is the smallest operator; round for unfixed arithmetic

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

2012-09-29

└ Chip area measurements (XC6VLX75T Virtex-6 FPGA)

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 unfixed arithmetic

- No additional comments

# 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

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

2012-09-29

└─ Architectural comparison to architectures generated by the Simulink HDL coder

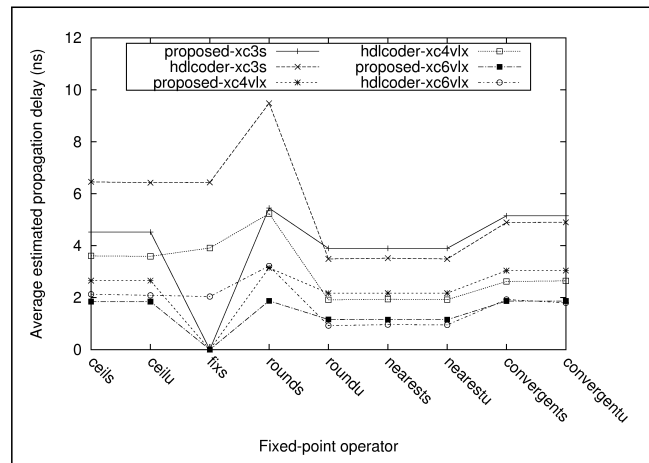
Architectural comparison to architectures generated by the Simulink HDL coder

- 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

- |                                                            |                                                  |
|------------------------------------------------------------|--------------------------------------------------|
| ■ <i>proposed</i>                                          | ■ <i>hdlcoder</i>                                |
| ■ Generic, portable descriptions                           | ■ Non-portable descriptions using bitwise tricks |
| ■ The same principles applicable to floating-point formats | ■ Not applicable to floating-point formats       |
| ■ No decrement operations needed                           | ■ Decrement operations                           |
| ■ No extended internal precision                           | ■ Need for an additional bit of accuracy         |

- Due to the architecture of modern FPGAs, reduction ORs prove beneficial due to the exploitation of hardwired, wide multiplexer primitives

# Speed measurements comparison to the Simulink HDL Coder



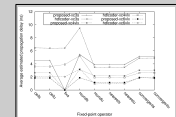
- The *proposed* *sfixed* units outperform the *hdlcoder* ones, by 31.3%, 27.7% and 27.6%, respectively
- Except *ceilu*, unsigned *hdlcoder* units are faster
- ☞ This is due to unsigned bitwise tricks applied by *hdlcoder* (e.g. *convergentu* uses a reduction OR gate)

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

2012-09-29

Speed measurements comparison to the Simulink HDL Coder

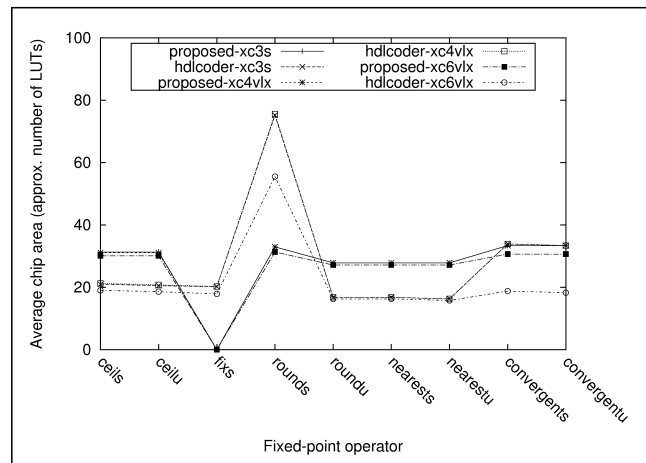
Speed measurements comparison to the Simulink HDL Coder



- The *proposed* *sfixed* 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)

- Results for all three architectures (Spartan-3, Virtex-4, Virtex-6)
- *hdlcoder* *nearestu* eliminates costlier post-processing by using an additional bit of internal accuracy

# Chip area measurements comparison to the Simulink HDL Coder



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

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

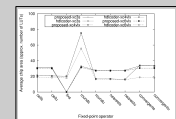
Design of fixed-point rounding operators for VHDL-2008

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

2012-09-29

└ Chip area measurements comparison to the Simulink HDL Coder

Chip area measurements comparison to the Simulink HDL Coder



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

- Results for all three architectures
- For the highly capacitive devices of nowadays, the area penalty to be paid is negligible and certainly justified when it is accompanied by speed improvements










# 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)

- 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)

- Future work regards the integration of the proposed fixed-point rounding units in a high-level synthesis prototype

# 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>

- No additional comments