Bibliography

[1] J. R. Armstrong, G. Frank, "Test Bench System Specification", February 1995 (Draft)

[2] S. Kottapalli, M.-W. Lin, J. R. Armstrong, "Test Plans for Testbench Generator Demonstration System", Research Report, October 1995

[3] G. A. Frank, J. R. Armstrong, W. R. Cyre, F. G. Gray, "TPALS: Test Planning Assistant for Legacy Systems", *RTI Proposal No. P816014*, Center for Digital Systems Engineering, Research Triangle Institute, February 1996

[4] R. G. Bennetts, *Design of Testable Logic Circuits*, Addison-Wesley Publishing, 1984

[5] W. C. Berg, R. D. Hess, "COMET: a Testability Analysis and Design Modification Package", *International Test Conference*, 1982, pp. 364-78

[6] J. Grason, "TMEAS - a Testability Measure Program", *Proc. 16th IEEE Design Automation Conf.*, 1979, pp. 156-161

[7] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983

[8] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, November, 1992

[9] [shaG94] G. A. Shaw, *RASSP Benchmark 1 - Technical Description - Synthetic Aperture Radar Processor*, Lincoln Laboratory, MIT, January, 1994

[10] Z. Xu, "Modeling SAR Signals and Sensors Using VHDL", *Master's Thesis*, Electrical Engineering Department, Virginia Tech, March, 1995

[11] J. C. Curlander, R. N. McDonough, *Synthetic Aperture Radar – Systems and Signal Processing*, John Wiley & Sons, Inc., 1991

[12] L. B. Jackson, *Digital Filters and Signal Processing*, 3rd Edition, Kluwer Academic Publishers, 1996

[13] J. R. Armstrong, F. G. Gray, *Structured Logic Design with VHDL*, Prentice Hall, Englewood Cliffs, NJ, 1993

[14] *IEEE Standard VHDL Language Reference Manual*, IEEE, New York, 1988

[15] J. R. Armstrong, "The RASSP Test Bench Generator", February, 1996

[16] M. A. Richards, "The RASSP Program: Overview and Accomplishments", Proceedings of First Annual RASSP Conference, Arlington VA, August, 1994

[17] R. Gummadi, "Methodology for structured VHDL model development", *Master's Thesis*, Electrical Engineering Department, Virginia Tech, April, 1995

[18] Comdisco Systems, INC., *Signal Processing WorkSystem - The DSP Framework*, 1992

[19] Synopsys, *Synopsys Graphical Environment User Guide*, Version 3.0, December, 1992

[20] J. C. Henry, "The Lincoln Laboratory 35 GHz Airborne SAR Imaging Radar System", *1991 National Telesystems Conference*, pp. 20-27, 1991

[21] H. Jensen, "Side-looking Airborne Radar"

[22] J. P. Fitch, "Generating Radar Images of the Earth with S-1 Computers"

[23] C. Elachi, "Radar Images of the Earth from Space"

[24] J. C. Curlander, R. N. McDonough, *Synthetic Aperture Radar - Systems and Signal Processing*, Wiley Interscience, 1991

[25] R. G. Bennetts, C. M. Maunder, G. D. Robinson, "CAMELOT: a Computer-Aided Measure for Logic Testability", *IEE Proc.* Volume 128, Part E, No. 5, 1981, pp. 177-189

[26] E. Kit, *Software Testing in the Real World: Improving the Process*, Addison-Wesley, 1995

[27] Ascent Logic Corporation, *RDD-100 User's Guide*, Release 4

[28] B. Zuerndorfer, G. A. Shaw, "SAR Processing for RASSP Application", *Proceedings of the First Annual RASSP Conference*, Arlington, VA, August 15-18, 1994, pp. 253-268

[29] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990

[30] P. C. Ward, J. R. Armstrong, "Behavioral Fault Simulation in VHDL", *27th ACM/IEEE Design Automation Conference*, Orlando, Florida, 1990

[31] M. D. O'Neill, D. D. Jani, C. H. Cho, J. R. Armstrong, "ETG: A Behavioral Test Generator", *Computer Hardware Description Languages and their Application*, edited by J. A. Darringer and F. J. Rammig, Elservier Science Publishers, 1990

[32] W. E. Howden, *Functional Program Testing and analysis*, McGraw-Hill, New York, NY, 1987

[33] IEEE Computer Society, *IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE Std. 729-1983

[34] B. Beizer, *Software System Testing and Quality Assurance*, Van Nostrand Reinhold, New York, NY, 1984

[35] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, CRC Press, Boca Raton, FL, 1995

[36] W. Hetzel, *The Complete Guide to Software Testing*, QED Information Sciences, Wellesley, Massachusetts, 1984

[37] B. Beizer, *Software Testing Techniques*, 2nd Edition, Van Nostrand Reinhold, New York, NY, 1990

[38] B. Beizer, *Black-Box Testing*, John Wiley & Sons, New York, NY, 1995

[39] T. C. Royer, *Software Testing Management: Life on the Critical Path*, Prentice-Hall, Englewood Cliffs, NJ, 1993

[40] C. Kaner, J. Falk, H. Q. Nguyan, *Testing Computer Software*, Van Nostrand Reinhold, New York, NY, 1993

[41] W. Miller, *The Engineering of Numerical Software*, Prentice-Hall, Englewood Cliffs, NJ, 1984

[42] R. A. DeMillo, W. M. McCracken, R. J. Martin, J. F. Passafiume, *Software Testing and Evaluation*, Benjamin/Cummings Publishing, 1987

[43] C.-K. Cho, *An Introduction to Software Quality Control*, John Wiley & Sons, 1980

[44] C.-K. Cho, *Quality Programming: Developing and Testing Software Using Statistical Quality Control*, John Wiley & Sons, 1987

[45] D. Smith, *HDL Chip Design: A Practical Guide for Designing, Synthesizing, and Simulating ASICs and FPGAs using VHDL or Verilog*, Doone Publications, 1996, Madison AL

[46] P. Walsh, D. Hoffman, "Automated Behavioral Testing of VHDL Components", *Canadian Conference on Electrical and Computer Engineering*, Calgary, Alberta, Canada, May, 1996

[47] M.-W. Lin, J. R. Armstrong, "Test Planning for VHDL DSP Models", *IEEE European Test Workshop (ETW'97)*, Cagliari, Italy, May, 1997

[48] M.-W. Lin, J. R. Armstrong, G. A. Frank, L. Concha, "A Functional Test Planning System for Validation of DSP Circuits Modeled in VHDL", to appear in *Proceedings of Spring Conference of VHDL International Users Forum*, Santa Clara, CA, March, 1998

[49] *Proceedings of European Design and Test Conference*, 1993

[50] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, 1979

[51] J. E. Heiser, "An Overview of Software Engineering", *Proceedings of 1997 IEEE AUTOTESTCON*, Anaheim, CA, Sep. 1997

[52] J. R. Armstrong, G. A. Frank, F. G. Gray, "Efficient Approaches to Testing VHDL DSP Models", *Journal on VSLI Signal Processing*, Vol. 14, November 1996, pp. 221-234

[53] J. R. Armstrong, F. G. Gray, M.-W. Lin, "VHDL Modeling and Model Testing for DSP Application", accepted by *IEEE Transactions on Industrial Electronics*

[54] S. Hrishikesh, "Behavioral Test Bench Development For DSP Models", *Masters Thesis*, Bradley Department of Electrical Engineering, Virginia Tech, Blacksburg, VA, March 1995

[55] P. Gowrisankaran, “Structural Test Bench Development For DSP Models”, *Masters Thesis*, Bradley Department of Electrical Engineering, Virginia Tech, Blacksburg, VA, March 1995

[56] R. A. DeMillo, R. J. Lipton, F. G. Sayward, “Hints on Test Data Selection: Help for the Practicing Programming”, *Computer*, Vol. 11(4) April 1978, pp.34-41

[57] S. Kottapalli, “A Test Plan Driven Test Bench Generation System”, *Masters Thesis*, Bradley Department of Electrical Engineering, Virginia Tech, Blacksburg, VA, May 1996

[58] J. A. Wicks, Jr. And J. R. Armstrong, “VHDL Model Efficiency”, *The Third Asia Pacific Conference on Hardware Description Languages (APCHDL '96)*, pp. 150-154

[59] *Synopsys VHDL System Simulator - Command Reference Manual*, Version 3.3b, Sept. 1995

[60] G. W. Jones, *Software Engineering*, John Wiley & Sons, 1990

[61] D. Heller, P. M. Brennan, *Motif Programming Manual*, Volume A and B, O’Reilly & Associates, 1994

[62] The Math Works Inc., *MATLAB Reference Guide*, Oct., 1992

[63] T. V. Raalte, *Xview Programming Manual*, O’Reilly & Associates, 1993

[64] R. A. DeMillo, E. Spafford, “The Mothra Software Testing Environment”, *Proceedings of 11th Software Engineering Workshop*, 1986

[65] A. J. Offutt, “A Practical System for Mutation Testing: Help for the Common Programmer”, *IEEE International Test Conference*, 1994

[66] M. A. Ould, C. Unwin, *Testing in Software Development*, Cambridge University Press, London, 1986

[67] W. Hetzel, “Principles of Computer Program Testing”, *Program Test Method*, Prentice-Hall, 1973, pp. 17-28

[68] W. E. Howden, “Functional Program Testing”, *IEEE Transactions on Software Engineering*, Vol. SE-6, March 1980, pp. 162-169

[69] W. J. Cody, W. Waite, *Software Manual for the Elementary Functions*, Prentice Hall, Englewood Cliffs, NJ, 1980

[70] K. Thearling, J. Abraham, "An Easily Computed Functional Level Testability Measure", *International Test Conference*, 1989, pp. 381-390

[71] S. Takasaki, M. Kawai, S. Funatsu, A. Yamada, "A Calculus of Testability Measure at the Functional Level", *IEEE Test Conference*, 1981, pp.95-101

[72] D. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", *IEEE Transactions on Software Engineering*, Vol. 23, No. 7, July 1997, pp.437-444

[73] T. J. Ostrand, M. J. Balcer, "Machine-Aided Production of Software Tests", *Fifth Annual Pacific Northwest Software Quality Conference*, Portland, OR, 1987, pp. 371-85

[74] R. Vemuri, P. Mamtora, P. Sinha, N. Kumar, J. Roy, R. Vutukuru, "Experiences in Functional Validation of a High Level Synthetic System", *30th Design Automation Conference*, 1993, pp. 194-201

[75] M. Ramachandran, "Requirements-Driven Software Test: A Process-Oriented Approach", *Software Engineering Notes*, Vol. 21, No. 4, July 1996, pp. 66-70

[76] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", *IBM J. Res. Develop.*, Vol. 10, July 1966, pp. 278-291

[77] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Trans. Comput.*, Vol. C-30, March 1981, pp. 215-222

[78] H. Fujiwara, T. Shimono, "On the Acceleration of Test Generation Algorithms", *IEEE Trans. Comput.*, Vol. C-32, December 1983, pp. 1137-1141

[79] C. H. Cho, J. R. Armstrong, "The B Algorithm: A Behavioral Test Generation Algorithm", *Proc. International Test Conference*, 1994, pp. 968-979

[80] P. Ward and J. R. Armstrong, "Behavioral Fault Simulation in VHDL", *Proc. 26th Design Automation Conference, June 1989*, pp. 587-593

[82] Y. H. Levendel, P. R. Menon, "Test Generation Algorithm for Computer Hardware Description Languages", *IEEE Trans. Comput.*, Vol c-31, July 1982, pp. 577-588

[82] S. Kapoor, J. R. Armstrong, S. R. Rao, "An Automatic Test Bench Generation System", *Proc. VHDL International Users Forum*, 1994, pp. 8-17

[83] W. Li, J. R. Armstrong, "Behavioral Test Generation with Fault Coverage Enhancement", *The Second Workshop on Hierarchical Test Generation*, Duisburg, Germany, September 1995

[84] P. N. Anirudhan, P. R. Menon, "Symbolic Test Generation for Hierarchically Modeled Digital System", *Proc. Int. Test Conf.*, 1989, pp. 461-469

[85] W. V. Quine, "A Way to Simplify Truth Functions", *Am. Math. Monthly*, vol. 62, no. 9, November, 1955, pp. 627-631

[86] E. J. McCluskey and H. Schorr, "Essential Multiple-Output Prime Implicants, Mathematical Theory of Automata", *Proc. Polytech. Inst. Brooklyn Symp.*, vol. 12, pp. 437-457, April, 1962

Appendix: VHDL Code Listing

Chirp

```
--FUNCTION:  Generate chirp signal

use work.MATH_REAL.all;
entity CHIRP is
   generic ( bw :                REAL;
             samp_freq :         REAL;
             pw :                REAL;
             prf :               REAL;
             NUM_TARGETS:        INTEGER;
             M :                 REAL_ARRAY );
   port ( ENABLE:      in BIT;
          N :          in REAL;
          R :          out REAL_ARRAY;
          I :          out REAL_ARRAY );
end CHIRP;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of CHIRP is
   begin
      process
         variable k:       REAL := 0.0;
         variable rv:      REAL_ARRAY(1 to M'high) := (others => 0.0);
         variable iv:      REAL_ARRAY(1 to M'high) := (others => 0.0);
      begin
         wait on N until ENABLE = '1';
         for i in 1 to NUM_TARGETS loop
           k := bw / pw;
           rv(i) := cos (  MATH_PI * k * (N-M(i)) * (N-M(i)) / (samp_freq * samp_freq) );
           iv(i) := sin (  MATH_PI * k * (N-M(i)) * (N-M(i)) / (samp_freq * samp_freq) );
         end loop;
         R <= rv;
         I <= iv;
```

```
      end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of CHIRP is
   begin
      process
         variable k:        REAL := 0.0;
         variable phase:    REAL := 0.0;
         variable rv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
         variable iv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
      begin
         wait on N until ENABLE = '1';
         for i in 1 to NUM_TARGETS loop
           k := bw / pw;
           phase := MATH_PI * k * (N-M(i)) * (N-M(i)) / (samp_freq * samp_freq);
           rv(i) := cos (  phase );
           iv(i) := sin (  phase );
         end loop;
         R <= rv;
         I <= iv;
      end process;
end BEHAVIOR2;
```

Complex Array Multiplier

```
--FUNCTION:  complex array multiplexer

work.MATH_REAL.all;
entity COMPARRMUL is
   generic ( NUM_TARGETS: INTEGER);
   port ( ENABLE: in BIT;
          Rin1: in REAL_ARRAY ;
          Iin1: in REAL_ARRAY ;
          Rin2: in REAL_ARRAY ;
          Iin2: in REAL_ARRAY ;
          Rout: out REAL_ARRAY ;
          Iout: out REAL_ARRAY );
end COMPARRMUL;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of COMPARRMUL is
```

```
   begin
      process
         variable rv1: REAL := 0.0;
         variable rv2: REAL := 0.0;
         variable iv1: REAL := 0.0;
         variable iv2: REAL := 0.0;
         variable RV:  REAL_ARRAY(1 to Rin1'high) ;
         variable IV:  REAL_ARRAY(1 to Rin1'high) ;
      begin
         wait on Rin1, Iin1, Rin2, Iin2 until ENABLE = '1';
         for i in 1 to NUM_TARGETS loop
            if Rin1(i) < -1.0e+36 then
               rv1 := 0.0;
            else
               rv1 := Rin1(i);
            end if;
            if Rin2(i) < -1.0e+36 then
               rv2 := 0.0;
            else
               rv2 := Rin2(i);
            end if;
            if Iin1(i) < -1.0e+36 then
               iv1 := 0.0;
            else
               iv1 := Iin1(i);
            end if;
            if Iin2(i) < -1.0e+36 then
               iv2 := 0.0;
            else
               iv2 := Iin2(i);
            end if;
            RV(i) := rv1 * rv2 - iv1 * iv2;
            IV(i) := rv1 * iv2 + rv2 * iv1;
         end loop;
         Rout <= RV;
         Iout <= IV;
      end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of COMPARRMUL is
   begin
      process
```

```
        variable rv1: REAL := 0.0;
        variable rv2: REAL := 0.0;
        variable iv1: REAL := 0.0;
        variable iv2: REAL := 0.0;
        variable RV:  REAL_ARRAY(1 to Rin1'high) ;
        variable IV:  REAL_ARRAY(1 to Rin1'high) ;
      begin
        wait on Rin1, Iin1, Rin2, Iin2 until ENABLE = '1';
        for i in 1 to NUM_TARGETS loop
          rv1 := Rin1(i);
          rv2 := Rin2(i);
          iv1 := Iin1(i);
          iv2 := Iin2(i);
          if rv1 < -1.0e+36 then
            rv1 := 0.0;
          end if;
          if rv2 < -1.0e+36 then
            rv2 := 0.0;
          end if;
          if iv1 < -1.0e+36 then
            iv1 := 0.0;
          end if;
          if iv2 < -1.0e+36 then
            iv2 := 0.0;
          end if;
          RV(i) := rv1 * rv2 - iv1 * iv2;
          IV(i) := rv1 * iv2 + rv2 * iv1;
        end loop;
        Rout <= RV;
        Iout <= IV;
      end process;
end BEHAVIOR2;
```

Compftp

--FUNCTION: Generate the complex tone--the carrier frequency part.

```
use work.MATH_REAL.all;
entity COMPTFP is
   generic ( freq:           REAL;
             samp_freq:      REAL;
```

```
            NUM_TARGETS:     INTEGER;
            M:               REAL_ARRAY );
   port ( ENABLE:     in BIT;
          N:          in REAL;
          R:          out REAL_ARRAY;
          I:          out REAL_ARRAY );
   end COMPTFP;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of COMPTFP is
   begin
      process
      variable phase: REAL;
         variable rv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
         variable iv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
      begin
         wait on N until ENABLE = '1';
         for i in 1 to NUM_TARGETS loop
            phase := 2.0 * MATH_PI * freq * (N-M(i)) / samp_freq ;
            rv(i) := cos (phase);
            iv(i) := sin (phase);
         end loop;
         R <= rv;
         I <= iv;
      end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of COMPTFP is
   begin
      process
         variable rv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
         variable iv:       REAL_ARRAY(1 to M'high) := (others => 0.0);
      begin
         wait on N until ENABLE = '1';
         for i in 1 to NUM_TARGETS loop
            rv(i) := cos (2.0 * MATH_PI * freq * (N-M(i)) / samp_freq );
            iv(i) := sin (2.0 * MATH_PI * freq * (N-M(i)) / samp_freq );
         end loop;
         R <= rv;
         I <= iv;
      end process;
end BEHAVIOR2;
```

Blocker

```
--FUNCTION:  BLOCK the signal outside the time span of chirp pulse

use work.MATH_REAL.all;
entity BLOCKER is
   generic ( samp_freq:       REAL;
             pw:              REAL;
             NUM_TARGETS:     INTEGER;
             M:               REAL_ARRAY );
   port ( ENABLE:     in BIT;
          Rin:        in REAL_ARRAY;
          Iin:        in REAL_ARRAY;
          N:          in REAL;
          Rout:       out REAL_ARRAY;
          Iout:       out REAL_ARRAY );
   end BLOCKER;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of BLOCKER is
 begin
      process
         variable RV:  REAL_ARRAY(1 to M'high) := (others =>0.0);
         variable IV:  REAL_ARRAY(1 to M'high) := (others =>0.0);
         begin
           wait on N until ENABLE = '1';
           for i in 1 to NUM_TARGETS loop
             if ( N < M(i) or (N-M(i)) / samp_freq > pw ) then
                RV(i) := 0.0;
                IV(i) := 0.0;
             else
                RV(i) := Rin(i);
                IV(i) := Iin(i);
             end if;
           end loop;
           Rout <= RV;
           Iout <= IV;
         end process;
 end BEHAVIOR1;
```

```
architecture BEHAVIOR2 of BLOCKER is
 begin
      process
         variable RV:  REAL_ARRAY(1 to M'high) := (others =>0.0);
         variable IV:  REAL_ARRAY(1 to M'high) := (others =>0.0);
         variable I:   INTEGER;
         begin
          wait on N until ENABLE = '1';
          I := 1;
          while I <= NUM_TARGETS loop
           if ( N < M(i) or (N-M(i)) / samp_freq > pw ) then
              RV(i) := 0.0;
              IV(i) := 0.0;
           else
              RV(i) := Rin(i);
              IV(i) := Iin(i);
           end if;
           I := I + 1;
          end loop;
          Rout <= RV;
          Iout <= IV;
         end process;
 end BEHAVIOR2;
```

Merger

```
--FUNCTION: Merge complex signals

use work.MATH_REAL.all;
entity MERGER is
   generic (NUM: INTEGER);
   port ( ENABLE:      in BIT;
          Rin:         in REAL_ARRAY;
          Iin:         in REAL_ARRAY;
          Rout:        out REAL:= 0.0;
          Iout:        out REAL:= 0.0);
   end MERGER;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of MERGER is
   begin
```

```
      process
         variable rv: REAL := 0.0;
         variable iv: REAL := 0.0;
      begin
         wait on Rin, Iin until ENABLE = '1';
         rv:=0.0;
         iv:=0.0;
         for i in 1 to NUM loop
            if Rin(i) > -1.0e+36 then
               rv := rv + Rin(i);
            end if;
            if Iin(i) > -1.0e+36 then
               iv := iv + Iin(i);
            end if;
         end loop;
         Rout <= rv;
         Iout <= iv;
      end process;
  end BEHAVIOR1;

architecture BEHAVIOR2 of MERGER is
    begin
      process
         variable rv: REAL := 0.0;
         variable iv: REAL := 0.0;
      begin
         wait on Rin, Iin until ENABLE = '1';
         rv:=0.0;
         iv:=0.0;
         for i in 1 to NUM loop
            if Rin(i) > -1.0e+36 then
               rv := rv + Rin(i);
            end if;
         end loop;
         for i in 1 to NUM loop
            if Iin(i) > -1.0e+36 then
               iv := iv + Iin(i);
            end if;
         end loop;
         Rout <= rv;
         Iout <= iv;
      end process;
```

```
end BEHAVIOR2;
```

Complex Multiplier

```
--FUNCTION:  complex multiplexer

use work.MATH_REAL.all;
entity COMPMUL is
   port ( ENABLE:      in BIT;
          Rin1:        in REAL := 0.0;
          Iin1:        in REAL := 0.0;
          Rin2:        in REAL := 0.0;
          Iin2:        in REAL := 0.0;
          Rout:        out REAL:= 0.0;
          Iout:        out REAL:= 0.0);
end COMPMUL;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of COMPMUL is
   begin
      process
         variable rv1: REAL := 0.0;
         variable rv2: REAL := 0.0;
         variable iv1: REAL := 0.0;
         variable iv2: REAL := 0.0;
      begin
         wait on Rin1, Iin1, Rin2, Iin2 until ENABLE = '1';
         if Rin1 < -1.0e+36 then
            rv1 := 0.0;
         else
            rv1 := Rin1;
         end if;
         if Rin2 < -1.0e+36 then
            rv2 := 0.0;
         else
            rv2 := Rin2;
         end if;
         if Iin1 < -1.0e+36 then
            iv1 := 0.0;
         else
            iv1 := Iin1;
```

```
            end if;
            if Iin2 < -1.0e+36 then
               iv2 := 0.0;
            else
               iv2 := Iin2;
            end if;
            Rout <= rv1 * rv2 - iv1 * iv2;
            Iout <= rv1 * iv2 + rv2 * iv1;
        end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of COMPMUL is
    begin
        process
            variable rv1: REAL := 0.0;
            variable rv2: REAL := 0.0;
            variable iv1: REAL := 0.0;
            variable iv2: REAL := 0.0;
        begin
            wait on Rin1, Iin1, Rin2, Iin2 until ENABLE = '1';
            rv1 := Rin1;
            rv2 := Rin2;
            iv1 := Iin1;
            iv2 := Iin2;
            if rv1 < -1.0e+36 then
              rv1 := 0.0;
            end if;
            if rv2 < -1.0e+36 then
              rv2 := 0.0;
            end if;
            if iv1 < -1.0e+36 then
              iv1 := 0.0;
            end if;
            if Iin2 < -1.0e+36 then
              iv2 := 0.0;
            end if;
            Rout <= rv1 * rv2 - iv1 * iv2;
            Iout <= rv1 * iv2 + rv2 * iv1;
        end process;
end BEHAVIOR2;
```

Negation

```
--FUNCTION:  Complex conjugate

use work.MATH_REAL.all;
entity NEG is
   port ( ENABLE:      in BIT;
          RX:          in REAL:=0.0;
          IX:          in REAL:=0.0;
          RY:          out REAL:=0.0;
          IY:          out REAL:=0.0 );
end NEG;

use work.MATH_REAL.all;
architecture BEHAVIOR1 of NEG is
   begin
      process
      begin
         wait on RX, IX until ENABLE = '1';
         RY <= RX;
         IY <= -IX;
      end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of NEG is
   begin
      process
      variable temp: REAL;
      begin
         wait on RX, IX until ENABLE = '1';
         RY <= RX;
         temp:= -1.0*IX;
         IY <= temp;
      end process;
end BEHAVIOR2;
```

Noise

```
--FUNCTION:  Generate the Gaussian noise

use work.MATH_REAL.all;
```

```
entity NOISE is
   generic ( SIGMA :       REAL );
   port (      ENABLE:     in BIT;
               N:          in  REAL;
               Rnoise:     out REAL;
               Inoise:     out REAL );
end NOISE;

architecture BEHAVIOR1 of NOISE is
   begin
      process
        variable U1, U2, MAG, PHASE: real;
        variable SEED1:  INTEGER := 7;
        variable SEED2:  INTEGER := 13;
      begin
        wait on N until ENABLE = '1';
        UNIFORM(SEED1, SEED2, U1);
        UNIFORM(SEED1, SEED2, U2);
        MAG := SQRT(-2.0*log(U1))*SIGMA;
        PHASE := 2.0*MATH_PI*U2;
        Rnoise <= MAG*cos(PHASE);
        UNIFORM(SEED1, SEED2, U1);
        UNIFORM(SEED1, SEED2, U2);
        MAG := SQRT(-2.0*log(U1));
        PHASE := 2.0*MATH_PI*U2;
        Inoise <= MAG*cos(PHASE);
      end process;
   end BEHAVIOR1;

architecture BEHAVIOR2 of NOISE is
   begin
      process
        variable U1, U2: real;
        variable SEED1:  INTEGER := 7;
        variable SEED2:  INTEGER := 13;
      begin
        wait on N until ENABLE = '1';
        UNIFORM(SEED1, SEED2, U1);
        UNIFORM(SEED1, SEED2, U2);
        Rnoise <= SQRT(-2.0*log(U1))*SIGMA*cos(2.0*MATH_PI*U2);
        UNIFORM(SEED1, SEED2, U1);
        UNIFORM(SEED1, SEED2, U2);
```

```
        Inoise <= SQRT(-2.0*log(U1))*SIGMA*cos(2.0*MATH_PI*U2);
      end process;
   end BEHAVIOR2;
```

SAR Return

```
--FUNCTION:  SAR return signal

use work.MATH_REAL.all;
entity SARRETURN is
   generic ( bw:              REAL;
             freq:            REAL;
             samp_freq:       REAL;
             pw:              REAL;
             prf:             REAL;
             NUM_TARGETS:     INTEGER;
             M:               REAL_ARRAY );
   port ( ENABLE:     in BIT;
          N:          in REAL;
          Rout:       out REAL;
          Iout:       out REAL );
end SARRETURN;

use work.MATH_REAL.all;
architecture STRUCTURE of SARRETURN is
   component COMPTFPC
   generic ( freq:                REAL;
             samp_freq:           REAL;
             NUM_TARGETS:         INTEGER;
             M:                   REAL_ARRAY );
   port ( ENABLE:     in BIT;
          N:          in REAL;
          R:          out REAL_ARRAY;
          I:          out REAL_ARRAY );
   end component;
   component CHIRPC
   generic ( bw :             REAL;
             samp_freq :      REAL;
             pw :             REAL;
             prf :            REAL;
             NUM_TARGETS:     INTEGER;
```

```
          M:                REAL_ARRAY );
port ( ENABLE:     in BIT;
       N:          in REAL;
       R:          out REAL_ARRAY;
       I:          out REAL_ARRAY );
end component;
component COMPARRMULC
   generic ( NUM_TARGETS: INTEGER);
   port (     ENABLE:          in BIT;
              Rin1:            in REAL_ARRAY;
              Iin1:            in REAL_ARRAY;
              Rin2:            in REAL_ARRAY;
              Iin2:            in REAL_ARRAY;
              Rout:            out REAL_ARRAY;
              Iout:            out REAL_ARRAY );
end component;
component BLOCKERC
generic ( samp_freq:          REAL;
          pw:                 REAL;
          NUM_TARGETS:   INTEGER;
       M:                     REAL_ARRAY );
 port ( ENABLE:    in BIT;
        Rin:       in REAL_ARRAY;
        Iin:       in REAL_ARRAY;
        N:         in REAL;
        Rout:      out REAL_ARRAY;
        Iout:      out REAL_ARRAY );
end component;
component MERGERC
   generic (NUM:      INTEGER);
   port (   ENABLE:  in BIT;
            Rin:        in REAL_ARRAY(1 to M'high);
            Iin:        in REAL_ARRAY(1 to M'high);
            Rout:       out REAL:= 0.0;
            Iout:       out REAL:= 0.0);
end component;
signal R1:  REAL_ARRAY(1 to M'high) := (others =>0.0);
signal I1:  REAL_ARRAY(1 to M'high) := (others =>0.0);
signal R2:  REAL_ARRAY(1 to M'high) := (others =>0.0);
signal I2:  REAL_ARRAY(1 to M'high) := (others =>0.0);
signal R3:  REAL_ARRAY(1 to M'high) := (others =>0.0);
signal I3:  REAL_ARRAY(1 to M'high) := (others =>0.0);
```

```
  signal R4:  REAL_ARRAY(1 to M'high) := (others =>0.0);
  signal I4:  REAL_ARRAY(1 to M'high) := (others =>0.0);
  begin
    C1: CHIRPC
      generic map( bw, samp_freq, pw, prf, NUM_TARGETS, M)
      port map(ENABLE, N, R1, I1);
    C2: COMPTFPC
      generic map( freq, samp_freq, NUM_TARGETS, M)
      port map(ENABLE, N, R2, I2);
    C3: COMPARRMULC
      generic map( NUM_TARGETS)
      port map(ENABLE, R1, I1, R2, I2, R3, I3);
    C4: BLOCKERC
      generic map( samp_freq, pw, NUM_TARGETS, M)
      port map(ENABLE, R3, I3, N, R4, I4);
    C5: MERGERC
      generic map(NUM_TARGETS)
      port map(ENABLE, R4, I4, Rout, Iout);
  end STRUCTURE;
```

Reference Signal

```
--FUNCTION:  SAR reference signal

use work.MATH_REAL.all;
entity REFERENCE is
  generic ( bw:          REAL;
            freq:        REAL;
            samp_freq:   REAL;
            pw:          REAL;
            prf:         REAL;
            Mr:          REAL_ARRAY(1 to 1));
  port ( ENABLE:    in BIT;
         N:         in REAL;
         Rout:      out REAL;
         Iout:      out REAL );
end REFERENCE;

use work.MATH_REAL.all;
architecture STRUCTURE of REFERENCE is
  component COMPTFPC
```

```
   generic ( freq:                REAL;
             samp_freq:           REAL;
             NUM_TARGETS:    INTEGER;
             M:                   in REAL_ARRAY );
   port ( ENABLE:   in BIT;
          N:            in REAL;
          R:            out REAL_ARRAY;
          I:            out REAL_ARRAY );
end component;
component CHIRPC
   generic ( bw:              REAL;
          samp_freq:          REAL;
          pw:                 REAL;
          prf:                REAL;
          NUM_TARGETS: INTEGER;
          M:                  REAL_ARRAY );
   port ( ENABLE:        in BIT;
          N:                 in REAL;
          R:                 out REAL_ARRAY;
          I:                 out REAL_ARRAY );
end component;
component COMPARRMULC
   generic ( NUM_TARGETS:    INTEGER );
   port (  ENABLE:        in BIT;
          Rin1:           in REAL_ARRAY;
          Iin1:           in REAL_ARRAY;
          Rin2:           in REAL_ARRAY;
          Iin2:           in REAL_ARRAY;
         Rout:            out REAL_ARRAY;
         Iout:            out REAL_ARRAY );
end component;
component MERGERC
   generic ( NUM:        INTEGER );
   port ( ENABLE:   in BIT;
          Rin:          in REAL_ARRAY(1 to 1);
          Iin:          in REAL_ARRAY(1 to Mr'high);
          Rout:        out REAL:= 0.0;
          Iout:        out REAL:= 0.0);
end component;
signal R1:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
signal I1:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
signal R2:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
```

```
   signal I2:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
   signal R3:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
   signal I3:  REAL_ARRAY(1 to Mr'high) := (others =>0.0);
   begin
      C1: CHIRPC
         generic map(bw, samp_freq, pw, prf, 1, Mr)
         port map(ENABLE, N, R1, I1);
      C2: COMPTFPC
         generic map(freq, samp_freq, 1, Mr)
         port map(ENABLE, N, R2, I2);
      C3: COMPARRMULC
         generic map(1)
         port map(ENABLE, R1, I1, R2, I2, R3, I3);
      C5: MERGERC
         generic map(1)
         port map(ENABLE, R3, I3, Rout, Iout);
   end STRUCTURE;
```

Deramp

```
--FUNCTION:  Deramper

use work.MATH_REAL.all;
entity DERAMP is
   port ( ENABLE:      in BIT;
          Rin1:        in REAL:=0.0;
          Iin1:        in REAL:=0.0;
          Rin2:        in REAL:=0.0;
          Iin2:        in REAL:=0.0;
          Rout:       out REAL:=0.0;
          Iout:       out REAL:=0.0 );
end DERAMP;

use work.MATH_REAL.all;
architecture STRUCTURE of DERAMP is
   component COMPMULC
      port ( ENABLE:      in BIT;
             Rin1:        in REAL;
             Iin1:        in REAL;
             Rin2:        in REAL;
             Iin2:        in REAL;
```

```
            Rout:          out REAL;
            Iout:          out REAL );
   end component;
   component NEGC
      port ( ENABLE:     in BIT;
            RX:          in REAL;
            IX:          in REAL;
            RY:          out REAL;
            IY:          out REAL );
   end component;
   signal R:   REAL:=0.0;
   signal I:   REAL:=0.0;
   begin
      C1: NEGC
         port map(ENABLE, Rin1, Iin1, R, I);
      C2: COMPMULC
         port map(ENABLE, R, I, Rin2, Iin2, Rout, Iout);
end STRUCTURE;
```

Sensor

```
use work.MATH_REAL.all;
use STD.TEXTIO.all;

entity SENSOR is
   generic ( bw_tx:              REAL;
             bw_rx:              REAL;
             carrier_freq:       REAL;
             samp_freq:          REAL;
             pw_tx:              REAL;
             pw_rx:              REAL;
             prf:                REAL;
             SIGMA:              REAL;
             NUM_TARGETS:        INTEGER;
             M:                  REAL_ARRAY);
   port ( ENABLE:     in BIT;
          N:          in REAL;
          Rout:       out REAL;
          Iout:       out REAL );
end SENSOR;
```

```
use work.MATH_REAL.all;
architecture STRUCTURE of SENSOR is
   component SARRETURNC
      generic ( bw:            REAL;
                freq:          REAL;
                samp_freq:     REAL;
                pw:            REAL;
                prf:           REAL;
               NUM_TARGETS:  INTEGER;
               M:             REAL_ARRAY );
      port ( ENABLE:         in BIT;
             N:              in REAL;
             Rout:           out REAL;
             Iout:           out REAL );
   end component;
   component REFERENCEC
   generic ( bw:           REAL;
             freq:         REAL;
             samp_freq:  REAL;
             pw:           REAL;
             prf:          REAL;
            Mr:            in REAL_ARRAY );
   port ( ENABLE:      in BIT;
          N:           in REAL;
          Rout:       out REAL;
          Iout:       out REAL );
   end component;
   component NOISEC
      generic ( SIGMA : REAL );
      port ( ENABLE: in BIT;
             N     :      in REAL;
             Rnoise:    out REAL;
             Inoise:     out REAL );
   end component;
   component MERGERC
      generic ( NUM:  INTEGER );
      port ( ENABLE:  in BIT;
           Rin:      in REAL_ARRAY;
           Iin:      in REAL_ARRAY;
           Rout:    out REAL:= 0.0;
           Iout:    out REAL:= 0.0);
   end component;
```

```
   component DERAMPC
      port ( ENABLE:      in BIT;
             Rin1:        in REAL;
             Iin1:        in REAL;
             Rin2:        in REAL;
             Iin2:        in REAL;
             Rout:        out REAL;
             Iout:        out REAL );
   end component;
   signal R1:  REAL_ARRAY(1 to 2) := (others => 0.0);
   signal I1:  REAL_ARRAY(1 to 2) := (others => 0.0);
   signal R2:  REAL := 0.0;
   signal I2:  REAL := 0.0;
   signal R3:  REAL := 0.0;
   signal I3:  REAL := 0.0;
   signal TWO: INTEGER := 2;
   begin
      CSARRETURNC: SARRETURNC
           generic map(bw_tx, carrier_freq, samp_freq, pw_tx, prf, NUM_TARGETS, M)
           port map(ENABLE, N, R1(1), I1(1));
      CNOISEC: NOISEC
           generic map(SIGMA)
           port map(ENABLE, N, R1(2), I1(2));
      CMERGERC: MERGERC
           generic map(2)
           port map(ENABLE, R1, I1, R2, I2);
      CREFERENCEC: REFERENCEC
           generic map(bw_tx, carrier_freq, samp_freq, pw_tx, prf, (1=> 0.0))
           port map(ENABLE, N, R3, I3);
      CDERAMPC: DERAMPC
         port map(ENABLE, R2, I2, R3, I3, Rout, Iout);
end STRUCTURE;
```

Comparator

```
use STD.textio.all;

entity COMPARATOR is
  port(ENABLE: in BIT);
end COMPARATOR;
```

```
architecture BEHAVIOR1 of COMPARATOR is
   begin
      process(ENABLE)
         constant MAXTARGETS: INTEGER := 100;
         type TARGETARRAY is array (1 to MAXTARGETS) of REAL;
         variable VLINE: LINE;
         file GOLDDAT: TEXT is in "gold.dat";
         file BINREPORT: TEXT is in "binreport";
         file GONOGO: TEXT is out "gonogo";
         variable GOLD: TARGETARRAY;
         variable DETECTED: TARGETARRAY;
         variable NGOLD: INTEGER;
         variable NDETECTED: INTEGER:= 0;
         variable BIN: INTEGER;
         variable MSR: REAL:=0.0;
         variable RESULT: INTEGER:=0;
         begin
          if ENABLE = '1' then
           READLINE(GOLDDAT, VLINE);
           READ(VLINE, NGOLD);
           for I in 1 to NGOLD loop
              READLINE(GOLDDAT, VLINE);
              READ(VLINE, GOLD(I));
           end loop;
           while not ENDFILE(BINREPORT) loop
                NDETECTED:= NDETECTED + 1;
                READLINE(BINREPORT, VLINE);
                READ(VLINE, BIN);
                READ(VLINE, DETECTED(NDETECTED));
           end loop;
           if NDETECTED = NGOLD then
             for I in 1 to NGOLD loop
                 MSR := MSR + (DETECTED(I) - GOLD(I)) * (DETECTED(I) -
GOLD(I));
             end loop;
             MSR := MSR / REAL(NGOLD);
             if MSR < 1.0 then
               RESULT := 1;
             end if;
           end if;
           WRITE(VLINE, RESULT);
           WRITELINE(GONOGO, VLINE);
```

```
            WRITE(VLINE, MSR);
            WRITELINE(GONOGO, VLINE);
          end if;
        end process;
end BEHAVIOR1;

architecture BEHAVIOR2 of COMPARATOR is
   begin
      process(ENABLE)
         constant MAXTARGETS: INTEGER := 100;
         type TARGETARRAY is array (1 to MAXTARGETS) of REAL;
         variable VLINE: LINE;
         file GOLDDAT: TEXT is in "gold.dat";
         file BINREPORT: TEXT is in "binreport";
         file GONOGO: TEXT is out "gonogo";
         variable GOLD: TARGETARRAY;
         variable DETECTED: TARGETARRAY;
         variable NGOLD: INTEGER;
         variable NDETECTED: INTEGER:= 0;
         variable BIN: INTEGER;
         variable MSR: REAL:=0.0;
         variable RESULT: INTEGER:=0;
         variable DIFF: REAL;
         begin
          if ENABLE = '1' then
            READLINE(GOLDDAT, VLINE);
            READ(VLINE, NGOLD);
            for I in 1 to NGOLD loop
               READLINE(GOLDDAT, VLINE);
               READ(VLINE, GOLD(I));
            end loop;
            while not ENDFILE(BINREPORT) loop
                 NDETECTED:= NDETECTED + 1;
                 READLINE(BINREPORT, VLINE);
                 READ(VLINE, BIN);
                 READ(VLINE, DETECTED(NDETECTED));
            end loop;
            if NDETECTED = NGOLD then
              for I in 1 to NGOLD loop
                 DIFF := DETECTED(I) - GOLD(I);
                 MSR := MSR + DIFF * DIFF;
              end loop;
```

```
            MSR := MSR / REAL(NGOLD);
            if MSR < 1.0 then
              RESULT := 1;
            end if;
          end if;
          WRITE(VLINE, RESULT);
          WRITELINE(GONOGO, VLINE);
          WRITE(VLINE, MSR);
          WRITELINE(GONOGO, VLINE);
        end if;
      end process;
end BEHAVIOR2;
```

SAR MUT

```
--use work.DSP_PRIMS1_PKG.all;
use work.DSP_PRIMS_PKG.all;
use work.TYPES_PKG.all;
use STD.textio.all;

package PRIMS is

   constant NSAMPLE: INTEGER := 4064;
   constant NIQ:     INTEGER := 2024;
   constant NRANGE:  INTEGER := 2048;

   type ARR1_RE_TYP is array ( POSITIVE range <> ) of REAL;
   type PROC_TYP is array(1 to 1024) of DI1_TYP(1 to 2048);
   type PROC1_TYP is array(1 to 2048) of DI1_TYP(1 to 1024);
   type CONVS is array(1 to 31) of DI1_TYP(1 to 512);
   type CONVS2 is array(1 to 16) of ARR1_TYP(1 to 512);
   type FCONVS is array(1 to 16) of DI1_TYP(1 to 1024);
   type IMAG_ARR is array(1 to 2048) of DI1_TYP(1 to 512);
   type INP_ARRAY is array(POSITIVE range <>) of BIT_VECTOR(1 to 32);

   procedure TAYLOR(
             constant N: in INTEGER;
             variable W:out ARR1_TYP
              );

   procedure VBIQ(
```

```
                variable SAMPLE: in ARR1_TYP(1 to NSAMPLE);
                variable IQ: out DI1_TYP(1 to NIQ)
              );

   procedure VBIQ1(
                variable SAMPLE: in ARR1_TYP(1 to NSAMPLE);
                variable IQ: out DI1_TYP(1 to NIQ)
              );

   procedure RG_COMPR(
                  variable IQ: in DI1_TYP(1 to NIQ) ;
                  variable W: in ARR1_TYP(1 to NRANGE);
                  variable RG_ARRAY: inout DI1_TYP(1 to NRANGE));

   procedure RG_COMPR1(
                  variable IQ: in DI1_TYP(1 to NIQ) ;
                  variable W: in ARR1_TYP(1 to NRANGE);
                  variable RG_ARRAY: inout DI1_TYP(1 to NRANGE));

   procedure POSTFILTER(
             variable RG_ARRAY: in DI1_TYP(1 to NRANGE);
             variable THRESHOLD: in REAL
                   );

   procedure POSTFILTER1(
             variable RG_ARRAY: in DI1_TYP(1 to NRANGE);
             variable THRESHOLD: in REAL
                   );

end PRIMS;


package body PRIMS is

   constant  MATH_PI :  real := 3.14159_26535_89793_23846;

   procedure TAYLOR(constant N: in INTEGER;
                    variable W: out ARR1_TYP
                   ) is
      variable NBAR: INTEGER := 6;
      variable SLL: REAL := 40.0;
      variable UP: ARR0_TYP(0 to 5):=(others=> 1.0);
```

```
variable DOWN: ARR0_TYP(0 to 5):=(others=> 1.0);
variable F: ARR0_TYP(0 to 5);
variable M: INTEGER := 1;
variable P:INTEGER := 1;
variable XI: REAL;
variable TEMP: REAL;
variable K: INTEGER :=1;
constant B: REAL := 31.6228;
constant A: REAL := 1.32;
constant SP: REAL :=0.8887;
begin
   for M in 1 to NBAR-1 loop
      for P in 1 to NBAR-1 loop
      UP(M) := UP(M)* (1.0-SP*(REAL(M)**2)/(A**2+(REAL(P)-0.5)**2));
      end loop;
   end loop;
   for M in 1 to NBAR-1 loop
      for P in 1 to NBAR-1 loop
         if ( M /= P ) then
           DOWN(M) := DOWN(M)*(1.0 -(REAL(M)**2)/(REAL(P)**2));
         end if;
      end loop;
   end loop;
   for M in 1 to NBAR-1 loop
      F(M) := 0.5*(REAL(-1)**(M+1))*UP(M)/DOWN(M);
   end loop;
   for I in 1 to N loop
      XI := (REAL(I) - (REAL(N)+2.0)/2.0)/REAL(N);
      TEMP:= 0.0;
      for M in 1 to NBAR-1 loop
         TEMP := TEMP + F(M)*COS(2.0*MATH_PI*REAL(M)*XI);
      end loop;
      W(I) := 1.0 + 2.0*TEMP;
   end loop;
end TAYLOR;

procedure VBIQ(variable SAMPLE: in ARR1_TYP(1 to NSAMPLE);
            variable IQ: out DI1_TYP(1 to NIQ)) is
   variable SIGN: INTEGER;
   variable EVEN: ARR1_TYP(1 to NSAMPLE/2);
   variable ODD: ARR1_TYP(1 to NSAMPLE/2);
   variable TEMP: ARR1_TYP(1 to NSAMPLE/2);
```

```
   variable SUM1, SUM2:   REAL;
   variable EVEN_COEFF: ARR0_TYP(0 to 7) := (-0.021133,
                                0.055895,
                               -0.148449,
                                0.406139,
                                0.917516,
                               -0.067483,
                               -0.011912,
                                0.019827);
   variable DEN_COEFF: ARR1_TYP(1 to 2) :=  (0.0,
                               0.0);
   variable ODD_COEFF: ARR0_TYP(0 to 7) :=  (0.019827,
                              -0.011912,
                               -0.067483,
                               0.917516,
                               0.406139,
                              -0.148449,
                               0.055895,
                              -0.021133);
   begin
      SIGN := 1;
      for I in 1 to NSAMPLE/2 loop
         EVEN(I) := REAL(SIGN)*SAMPLE(I*2-1);
         ODD(I) := REAL(SIGN)*SAMPLE(I*2);
         SIGN := -SIGN;
      end loop;
      for I in 1 to NIQ loop
         SUM1 := 0.0;
         for m in 0 to 7 loop
            SUM1 := SUM1 + EVEN_COEFF(m) * EVEN(I+m);
         end loop;
         SUM2 := 0.0;
         for m in 0 to 7 loop
            SUM2 := SUM2 + ODD_COEFF(m) * ODD(I+m);
         end loop;
         IQ(I) := (SUM1, SUM2);
      end loop;
end VBIQ;

procedure VBIQ1(variable SAMPLE: in ARR1_TYP(1 to NSAMPLE);
           variable IQ: out DI1_TYP(1 to NIQ)) is
   variable SIGN: INTEGER;
```

```
variable EVEN: ARR1_TYP(1 to NSAMPLE/2);
variable ODD: ARR1_TYP(1 to NSAMPLE/2);
variable TEMP: ARR1_TYP(1 to NSAMPLE/2);
variable SUM1, SUM2:   REAL;
variable EVEN_COEFF: ARR0_TYP(0 to 7) := (-0.021133,
                                 0.055895,
                                -0.148449,
                                 0.406139,
                                 0.917516,
                                -0.067483,
                                -0.011912,
                                 0.019827);
variable DEN_COEFF: ARR1_TYP(1 to 2) :=  (0.0,
                                0.0);
variable ODD_COEFF: ARR0_TYP(0 to 7) :=  (0.019827,
                                -0.011912,
                                 -0.067483,
                                 0.917516,
                                 0.406139,
                                -0.148449,
                                 0.055895,
                                -0.021133);
begin
   for I in 1 to NSAMPLE/2 loop
      if I rem 2 = 1 then
        EVEN(I) := SAMPLE(I*2-1);
        ODD(I) := SAMPLE(I*2);
      else
       EVEN(I) := -SAMPLE(I*2-1);
       ODD(I) := -SAMPLE(I*2);
      end if;
   end loop;
   for I in 1 to NIQ loop
      SUM1 := 0.0;
      for m in 0 to 7 loop
         SUM1 := SUM1 + EVEN_COEFF(m) * EVEN(I+m);
      end loop;
      SUM2 := 0.0;
      for m in 0 to 7 loop
         SUM2 := SUM2 + ODD_COEFF(m) * ODD(I+m);
      end loop;
      IQ(I) := (SUM1, SUM2);
```

```
        end loop;
 end VBIQ1;

 procedure RG_COMPR(variable IQ: in DI1_TYP(1 to NIQ) ;
                    variable W: in ARR1_TYP(1 to NRANGE);
                    variable RG_ARRAY: inout DI1_TYP(1 to NRANGE)) is
    variable TEMP: DI1_TYP(1 to NRANGE);
    variable FORW: INTEGER := -1;
    variable SIGN: INTEGER;
    begin
       SIGN:=1;
       for I in 1 to NIQ loop
          TEMP(I) := IQ(I)*REAL(SIGN)*W(I);
          SIGN:=-SIGN;
       end loop;
       for I in NIQ + 1 to NRANGE loop
         TEMP(I) := (0.0,0.0);
       end loop;
       FFT1(TEMP, FORW, RG_ARRAY);
 end RG_COMPR;

 procedure RG_COMPR1(variable IQ: in DI1_TYP(1 to NIQ) ;
                    variable W: in ARR1_TYP(1 to NRANGE);
                    variable RG_ARRAY: inout DI1_TYP(1 to NRANGE)) is
    variable TEMP: DI1_TYP(1 to NRANGE);
    variable FORW: INTEGER := -1;
    variable SIGN: INTEGER;
    begin
       for I in 1 to NIQ loop
          if I rem 2 = 1 then
            TEMP(I) := IQ(I)*W(I);
          else
            TEMP(I) := IQ(I)*W(I)*(-1.0);
          end if;
       end loop;
       for I in NIQ + 1 to NRANGE loop
          TEMP(I) := (0.0,0.0);
       end loop;
       FFT1(TEMP, FORW, RG_ARRAY);
 end RG_COMPR1;

procedure POSTFILTER(
```

```
   variable RG_ARRAY: in DI1_TYP(1 to NRANGE);
   variable THRESHOLD: in REAL
                ) is
   variable MAG: ARR1_RE_TYP( 1 to NRANGE);
   variable TEMP: REAL;
   file OUTVECT: TEXT is out "binreport";
   variable VLINE: LINE;
   begin
      for I in 1 to NRANGE loop
         TEMP:=RG_ARRAY(I)(0)*RG_ARRAY(I)(0)+RG_ARRAY(I)(1)*
         RG_ARRAY(I)(1);
         TEMP:=SQRT(TEMP);
         if (TEMP<THRESHOLD) then
           MAG(I):=0.0;
         else
           MAG(I):=TEMP;
         end if;
      end loop;

      for I in 2 to NRANGE - 1 loop
         if MAG(I) > MAG(I-1) and MAG(I) > MAG(I+1) then
           WRITE(VLINE, I);
           WRITE(VLINE, ' ');
           WRITE(VLINE, (REAL(I)-1004.0)*0.22871+7259.9);
           WRITELINE(OUTVECT, VLINE);
         end if;
      end loop;
   end POSTFILTER;

procedure POSTFILTER1(
   variable RG_ARRAY: in DI1_TYP(1 to NRANGE);
   variable THRESHOLD: in REAL
                ) is
   variable MAG: ARR1_RE_TYP(1 to NRANGE);
   variable TEMP: REAL;
   file OUTVECT: TEXT is out "binreport";
   variable VLINE: LINE;
   begin
      for I in 1 to NRANGE loop
      TEMP:=RG_ARRAY(I)(0)*RG_ARRAY(I)(0)+RG_ARRAY(I)(1)*
      RG_ARRAY(I)(1);
         TEMP:=SQRT(TEMP);
```

```
            if (TEMP>=THRESHOLD) then
              MAG(I):=TEMP;
            else
              MAG(I):=0.0;
            end if;
          end loop;
          for I in 2 to NRANGE - 1 loop
            if MAG(I) > MAG(I-1) and MAG(I) > MAG(I+1) then
              WRITE(VLINE, I);
              WRITE(VLINE, ' ');
              WRITE(VLINE, (REAL(I)-1004.0)*0.22871+7259.9);
              WRITELINE(OUTVECT, VLINE);
            end if;
          end loop;
        end POSTFILTER1;
end PRIMS;

--use work.DSP_PRIMS1_PKG.all;
use work.PRIMS.all;
use work.DSP_PRIMS_PKG.all;
use work.TYPES_PKG.all;
use STD.textio.all;

    entity MUT1D is
      port( ENABLE: in BIT);
    end MUT1D;

    architecture BEHAVIOR1 of MUT1D is
       begin
          process(ENABLE)
             variable TAYLW: ARR1_TYP(1 to NRANGE);
             variable VLINE: LINE;
             variable V: BIT_VECTOR(1 to 32);
             file INVECT: TEXT is in "raw.dat";
             file OUTVECT: TEXT is out "sar.dat";
             variable CHECK: BIT := '0';
             variable I: INTEGER := 1;
             variable K: INTEGER;
             variable SAMPLE: ARR1_TYP(1 to NSAMPLE);
             variable IQ: DI1_TYP(1 to NIQ);
             variable RG_ARRAY: DI1_TYP(1 to NRANGE);
             variable RNOT: REAL;
```

```
        variable MIN_THRESHOLD: REAL:=1000.0;
        variable MAX_THRESHOLD: REAL:=1000.0;
        variable THRESHOLD: REAL;
        begin
          if ENABLE = '1' then
            TAYLOR(NRANGE,TAYLW);
            for I in 1 to NSAMPLE loop
               READLINE(INVECT, VLINE);
               READ(VLINE, SAMPLE(I));
            end loop;
            VBIQ(SAMPLE, IQ);
            RG_COMPR(IQ, TAYLW, RG_ARRAY);
            for I in 1 to NRANGE loop
               WRITE(VLINE, RG_ARRAY(I)(0));
               WRITE(VLINE, ' ');
               WRITE(VLINE, RG_ARRAY(I)(1));
               WRITELINE(OUTVECT, VLINE);
            end loop;
            THRESHOLD:=0.5*(MIN_THRESHOLD+MAX_THRESHOLD);
            POSTFILTER(RG_ARRAY,THRESHOLD);
          end if;
      end process;
  end BEHAVIOR1;

architecture BEHAVIOR2 of MUT1D is
   begin
      process(ENABLE)
         variable TAYLW: ARR1_TYP(1 to NRANGE);
         variable VLINE: LINE;
         variable V: BIT_VECTOR(1 to 32);
         file INVECT: TEXT is in "raw.dat";
         file OUTVECT: TEXT is out "sar.dat";
         variable CHECK: BIT := '0';
         variable I: INTEGER := 1;
         variable K: INTEGER;
         variable SAMPLE: ARR1_TYP(1 to NSAMPLE);
         variable IQ: DI1_TYP(1 to NIQ);
         variable RG_ARRAY: DI1_TYP(1 to NRANGE);
         variable RNOT: REAL;
         variable MIN_THRESHOLD: REAL:=1000.0;
         variable MAX_THRESHOLD: REAL:=1000.0;
         variable THRESHOLD: REAL;
```

```
        begin
          if ENABLE = '1' then
            TAYLOR(NRANGE,TAYLW);
            for I in 1 to NSAMPLE loop
               READLINE(INVECT, VLINE);
               READ(VLINE, SAMPLE(I));
            end loop;
            VBIQ1(SAMPLE, IQ);
            RG_COMPR1(IQ, TAYLW, RG_ARRAY);
            for I in 1 to NRANGE loop
               WRITE(VLINE, RG_ARRAY(I)(0));
               WRITE(VLINE, ' ');
               WRITE(VLINE, RG_ARRAY(I)(1));
               WRITELINE(OUTVECT, VLINE);
            end loop;
            THRESHOLD:=(MIN_THRESHOLD+MAX_THRESHOLD)/2.0;
            POSTFILTER1(RG_ARRAY,THRESHOLD);
          end if;
      end process;
  end BEHAVIOR2;
```

Test Bench

```
use work.MATH_REAL.all;
use STD.TEXTIO.all;

entity TB is
port (  M:        in REAL_ARRAY(1 to 10);
        Mr:       in REAL_ARRAY(1 to 1) );
end TB;

use work.MATH_REAL.all;
architecture STRUCTURE of TB is
   component SENSORC
port ( ENABLE:        in BIT;
           N:           in REAL;
           Rout:         out REAL;
           Iout:         out REAL );
   end component;
   component MUT1DC
    port ( ENABLE: in BIT);
```

```
    end component;
    component COMPARATORC
      port ( ENABLE: in BIT );
    end component;
    signal S1:   BIT := '0';
    signal S2:   BIT := '0';
    signal S3:   BIT := '0';
    signal Rout: REAL := 0.0;
    signal Iout: REAL := 0.0;
    signal N:    REAL := 0.0;
    begin
       CSENSORC: SENSORC
          port map(S1, N, Rout, Iout);
       CMUT1DC: MUT1DC
            port map(S2);
       CCOMPARATORC: COMPARATORC
            port map(S3);
end STRUCTURE;
```

Vita

Morris Mengwei Lin

is also officially known as Morris Lin or Meng-Wei Lin. He was born in August of 1965 in Hualien, Taiwan. He graduated from Hualien High School in Hualien in June of 1984. He passed the nationwide Joint College Entrance Exam and ranked among top 100 among dozens of thousands of high school graduates in July of 1984. He then entered National Taiwan University in Taipei, Taiwan where he graduated with a Bachelor of Science degree in Electrical Engineering in June of 1988. He entered graduate school at University of Southern California in Los Angeles, CA in August of 1991 and graduated with a Master of Science degree in Computer Engineering in May of 1993. Finally, Morris entered Virginia Polytechnic Institute and State University in Blacksburg, VA in August of 1994 and graduated with a Ph.D. degree in Electrical Engineering in February of 1998.

Morris served in the Taiwan Army Signal Corps as a commissioned second lieutenant to fulfill his compulsory military service as a male citizen of Taiwan from July of 1988 to May of 1990. He completed the reserved officer training at the Army Academy of Telecommunications and Electronics in Taoyuan, Taiwan in November of 1988 and was appointed a software engineer and later a platoon leader. After discharged from the Signal Corps, he worked as a full-time teaching assistant in the Department of Computer Science, Soochow University in Taipei, Taiwan in the year of 1990-1991. Morris also worked as a graduate research assistant in the Virginia Tech Information Systems Center from January of 1995 to December of 1997. He has started his career as a hardware design engineer in the VLSI Technology Center of Hewlett-Packard Company in Fort Collins, Colorado since February of 1998.

Morris is an active member of the honor societies of ΦΚΦ, HKN, and ΦΒΔ. His research interests include testing, VLSI design, digital image compression, parallel algorithms, and CAD algorithms.