Customized Simulation Tool for Mixed Analog-Digital Integrated Circuits

R. Prakash Rao[1] and Dr. B.K. Madhavi[2]
[1]St. Peter's Engineering College, Near Forest Academy, Dulapally, Hyderabad, India
[2]Geetanjali College of Engineering and Technology, Cheryala, Hyderabad, India
prakashiits@gmail.com, bkmadhavi2009@gmail.com

***Abstract*– In the past few years, mixed analog-digital IC designs have become increasingly popular and the complexity of these chips has been growing. However, one of the major bottlenecks for achieving low cost and high quality design is the lack of simulation tools. Currently, designers need to manually split the design into analog and digital portions which are verified separately using analog and digital simulators respectively. There is no easy and robust way to verify the mixed analog-digital IC designs with one simulator. Hence, we are going to propose a new methodology to simulate a mixed analog-digital IC designs with one simulator. To achieve the proposed methodology we have defined the analog definitions in digital domain library. This library definition was integrated into the standard definition library, results in, we have provided a simple method to use HDL definition in EDA tool such as ModelSim PE 10.1b for analog also. As we are using a single tool for both analog and mixed signal design IC's the tool portability or optimization of the tool had been achieved for mixed analog-digital integrated circuits.**

***Keywords*–** Mixed Analog-Digital IC, Simulation Tools, Analog Definitions, Library Definition, Modelsim PE 10.1b and Tool Portability

I. INTRODUCTION

In the past, designers have used a variety of simulation methodologies to verify designs that contained both analog and digital circuits. At the very highest levels of abstraction, system designers have used C/C++ and Matlab to model systems that would be implemented with analog and digital circuits; but this approach usually doesn't try to represent any implementation issues. At the next level down in the hierarchy, designers have used Saber by Analogy and similar tools to model mixed-signal systems. At the lowest level of abstraction, designers have modeled all the analog and digital circuits at the transistor level and used Spice-like simulators, or reduced-complexity transistor-level simulators.

Designers are just beginning to use the VHDL and Verilog AMS languages and this approach fits somewhere in the middle compared to the above levels. The AMS extensions allow a designer to use VHDL or Verilog to describe analog circuits at different levels of abstraction, ranging from behavioral to structural. The AMS description is usually then translated to a netlist and simulated with a Spice-like simulator.

Another approach offered by major CAD companies is to provide a simulation environment that allows the user to choose from different levels of abstraction for a given simulation. Digital blocks are represented with an HDL and simulated with an HDL simulator; analog blocks are represented with transistors or an AMS HDL and simulated with a Spice-like simulator. A software backplane allows the HDL and Spice simulators to communicate via interprocess communication.

Typically lower levels of abstraction translate to slower simulation time. Consequently, simulating large mixed-signal designs solely at the transistor level with a standard Spice-like simulator may not be practical. The benefits of Spice simulation tools are that they provide the most detailed level of modeling and analysis including: DC, transient, small signal AC, and zero's/pole's of filters. The costs of Spice simulation are often long simulation times and tedious design entry.

To overcome the analog and mixed –signal designing issues a new approach in mixed-signal modeling and simulation is required at the lowest level of integration. A mixed signal level designing for digital modeling of the analog circuitry is to be developed for the modeling of analog circuitry in HDL environment.

Section II, proposed by the design. Section III explains the floating point arithmetic representations. In section IV we explained the complete ModelSim tool analysis and adding of the user defined library with the resource library. Section V and VI explains the case study and implementation respectively. Finally, section VII and VIII explains the results and conclusion.

II. PROPOSED DESIGN

Traditionally, if a system consists of both analog and digital systems on the same platform called mixed-signal systems, neither the analog tools nor the digital tools will support mixed signal designs. These mixed signal designs are used most-widely in DSP systems for audio and video purposes. So to simulate such mixed signal designs, presently the dedicated floating point arithmetic units are used in the CAD tools like XILINX new version ISE tool with the latest target devices like Vertex, Kintex etc. These dedicated floating point arithmetic units need to buy from different vendors. But, here we are proposing that after the extensive study of the ModelSim tool flow, instead of buying the

dedicated floating pint arithmetic units from different vendors, we have integrated the floating point arithmetic features to the ModelSim tool in which, earlier we would have simulated the fixed point numbers only. So that now the upgraded ModelSim tool can be used to perform the complete DSP operations like video and audio.

Since, the analog signal is decomposed or reconstructed with the signal samples and those signal samples could be defined with floating point values as shown in Fig. 1 and if these floating point features of analog signals are added to the resource library or standard library of any CAE tool, the particular tool will be upgraded with both the analog and digital features.

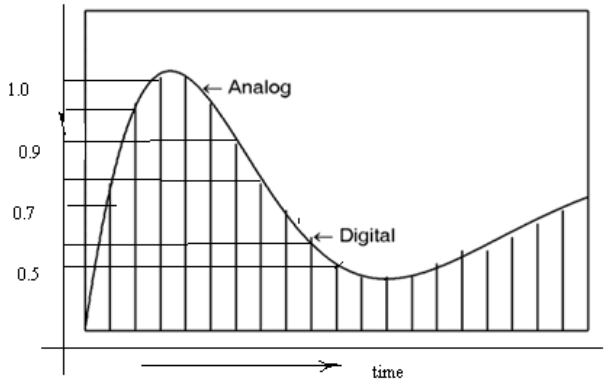

Fig. 1: Continuous wave signal with samples of 0.5, 0.7, 0.9, 1.0 (float values)

III. FLOATING POINT NUMBER REPRESENTATION

There are several ways to represent real numbers on computers. Fixed point places a radix point somewhere in the middle of the digits, and is equivalent to using integers that represent portions of some unit. For example, one might represent 1/100ths of a unit; if you have four decimal digits, you could represent 10.82, or 00.01. Another approach is to use rational, and represent every number as the ratio of two integers. Floating-point representation - the most common solution - basically represents reals in scientific notation [5]. Scientific notation represents numbers as a base number and an exponent.

For example, 123.456 could be represented as 1.23456×10^2. In hexadecimal, the number 123.abc might be represented as $1.23abc \times 16^2$. Floating-point solves a number of representation problems. Fixed-point has a fixed window of representation, which limits it from representing very large or very small numbers. Also, fixed-point is prone to a loss of precision when two large numbers are divided. Floating-point [1], on the other hand, employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease.

A. IEEE 754 Floating Point Standard

IEEE 754 floating point standard [5] is the most common representation today for real numbers on computers. The IEEE (Institute of Electrical and Electronics Engineers) has produced a Standard to define floating-point representation and arithmetic. The standard brought out by the IEEE come to be known as IEEE 754. The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely used standard for floating point computation, and is followed by many CPU and FPU implementations [2].

The standard defines formats for representing floating-point numbers including negative numbers and denormal numbers special values i.e. infinities and NANs together with a set of floating-point operations that operate on these values. It also specifies four rounding modes which are round to zero, round to nearest, round to infinity and round to even and five exceptions including when the exceptions occur, and what happens when they do occur. Dealing with fixed-point arithmetic will limit the usability of a processor. If operations on numbers with fractions (e.g., 10.2445), very small numbers (e.g., 0.000004), or very large numbers (e.g., 42.243x105) are required, then a different one representation is in order is the floating-point arithmetic [4].

IV. UPGRADING the MODELSIM TOOL

A. General

Since, ModelSim is the user friendly tool [11], in our work we have chosen ModelSim tool, and upgraded the features. ModelSim is a verification and simulation tool for VHDL, Verilog, SystemVerilog, SystemC, and mixed-language designs. Here, we are going to provide a brief conceptual overview of the ModelSim simulation environment.

B. Simulation Flow in ModelSim

The below Fig. 2 shows the basic steps for simulating a design in ModelSim.

Creating the Working Library: In ModelSim, all designs are compiled into a library. We start a new simulation in ModelSim by creating a working library called "work". "Work" is the library name used by the compiler as the default destination for compiled design units.

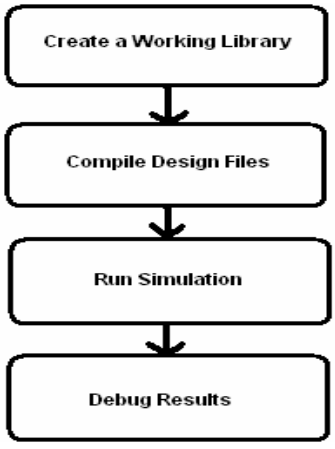

Fig. 2: Basic Simulation Flow Diagram

Compiling the Design: Before we simulate a design, we must first create a library and compile the source code into that library as given below.

(i) Create a new directory and copy the design files for this lesson into it. Start by creating a new directory for this exercise.

(ii) Start ModelSim if necessary.

- Type **vsim** at a UNIX shell prompt or use the ModelSim icon in Windows. Upon opening ModelSim for the first time, we will see the Welcome to ModelSim dialog. Click **Close**.
- Select **File > Change Directory** and change to the directory you created in step (i).

(iii) Create the working library.

- Select **File > New > Library**.This opens a dialog where you specify physical and logical names for the library. We can create a new library or map to an existing library. We will be doing the former.
- Type **work** in the Library Name field if it is not entered automatically Click **OK**.

ModelSim creates a directory called *work* and writes a specially formatted file named *_info* into that directory. The *_info* file must remain in the directory to distinguish it as a ModelSim library. Do not edit the folder contents from your operating system; all changes should be made from within ModelSim. ModelSim also adds the library to the list in the Workspace and records the library mapping for future reference in the ModelSim initialization file (*modelsim.ini*). When you pressed OK, several lines were printed to the Main window Transcript pane:

```
vlib work
vmap work work
# Copying C:\modeltech\win32/../modelsim.ini to modelsim.ini
# Modifying modelsim.ini
# ** Warning: Copied C:\modeltech\win32/../modelsim.ini to modelsim.ini.
# Updated modelsim.ini.
```

The first two lines are the command-line equivalent of the menu commands you invoked. Many menu-driven functions will echo their command-line equivalents in this fashion. The other lines notify you that the mapping has been recorded in a local ModelSim initialization file. After creating the working library, compile the design units into it. The ModelSim library format is compatible across all supported platforms. We can simulate your design on any platform without having to recompile your design. With the working library created, we are ready to compile your source files. We can compile by using the menus and dialogs of the graphic interface, as in the Verilog or VHDL.

V. CASE STUDY

The DWT represents a good example of the complexity that today's mixed signal integrated circuits present to the simulation tools. This complexity comes from the variety of functional features that must be supported by the design of Input FIFO, Buffer, High Pass, Low Pass Filters called Filter Bank, Coefficient Register, Controller, Energy Calculator, Energy RAM, Sample RAM. Fig. 3 shows a simplified block diagram of the chip. The input of speech signal will be converted as speech samples using MATLAB. These MATLAB coefficients which will be in the form of floating point, given as the input to the I/P FIFO. These floating point MATLAB coefficients will be processed using the IEEE-754 format.

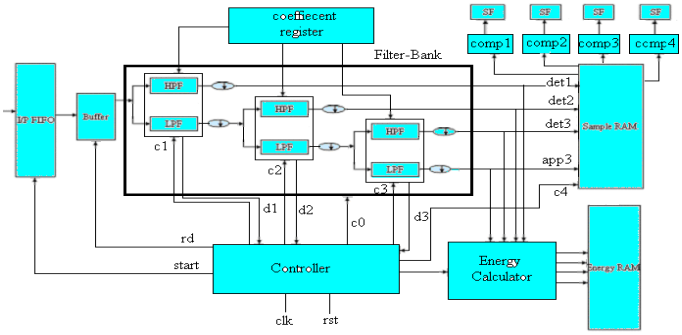

Fig. 3: Proposed system for DWT decomposer module

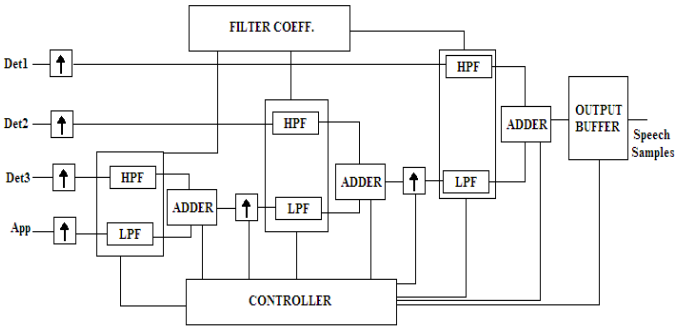

Fig. 4: Proposed system for DWT reconstructor module

Fig. 4 shows the proposed system for DWT reconstructor module. The signal applying at the input will be decomposed by the DWT decomposer and reconstructed by the reconstructor module and hence original signal could get at the output of the reconstructor module.

VI. IMPLEMENTATION

The proposed system is realized using VHDL language for it's functional definition. The HDL modeling is carried out in top-down approach with user defined package support for floating point operation and structural modeling for recursive implementation of the filter bank logic. For the realization a package is defined with user defined record data type as:

```
type real_single is
record
sign : std_logic;
```

```
exp: std_logic_vector(3 downto 0);
mantissa: std_logic_vector(10 downto 0);
end record;
```

The floating notation is implemented using 16 bit IEEE-754 standards as presented in below Fig. 6.

Sign. (1)	Exp. (4)	Mantissa (11)

Fig. 6: 16 bit IEEE-754 standard

The floating-point addition, multiplication and shifting operation are implemented as procedures in the user defined package and are repeatedly called in the implementation for recursive operation. The procedures are defined as:

```
procedure shifftl (arg1: std_logic_vector;arg2: integer;arg3
:out std_logic_vector);
procedure shifftr (a:in std_logic_vector; b:in integer;result:
out std_logic_vector);
procedure addfp (op1,op2: in real_single;op3: out
real_single) ;
procedure fpmult (op1,op2: in real_single;op3: out
real_single) ;
```

For performing the convolution operation, filter coefficients are defined as constant in this package and are called by name in filter implementation.

```
constant lpcof0: real_single:=('1',"0100","00001001000");
constant lpcof1: real_single:=('0',"0100","11001010111");
constant lpcof2:real_single:=('0',"0110","10101100010");
constant lpcof3:real_single:= ('0',"0101","11101110100");
constant hpcof0: real_single:= ('1',"0101","11101110100");
constant hpcof1:real_single:=('0',"0110","10101100010");
constant hpcof2:real_single:=('1',"0100","11001010111");
constant hpcof3:real_single:=('1',"0100","00001001000");
```

Using the above definitions the filters are designed for high pass and low pass operation. The recursive implementation is defined as:

```
for k in 1 downto 0 loop
old(k):=shift(k);
fpmult(old(k)(0),hpf(k+1),pro(k)(0));
proper(j,k):=pro(k)(0);
addfp(acc(k)(0),pro(k)(0),acc(k)(0));
acer(j,k):=acc(k)(0);
shift(k+1):=shift(k);
end loop;
```

For the evaluation of the implemented design the test vectors are passed through the test bench generated from Matlab tool. The continuous output is discretized using Matlab tool where each coefficient is converted to 16-bit floating notation and passed to the test bench for HDL interface. The coefficients obtained from the filter bank after convolution is then compared with the results obtained from the Matlab decomposition for accuracy evaluation.

```
library ieee;
use work.math_pack1.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;
entity topmodule_wb is
end topmodule_wb;
architecture TB_ARCHITECTURE of topmodule_wb is
component topmodule
port (
clk : in std_logic;
rst : in std_logic;
start : in std_logic;
read1 : in std_logic);
end component;
signal STIM_clk : std_logic;
signal TMP_clk : std_logic;
signal STIM_rst : std_logic;
signal STIM_start : std_logic;
signal STIM_read1 : std_logic;
signal WPL : WAVES_PORT_LIST;
signal TAG : WAVES_TAG;
signal ERR_STATUS: STD_LOGIC:='L';
begin
CLOCK_GEN_FOR_clk: process
begin
if END_SIM = FALSE then
TMP_clk <= '0';
wait for 50 ns;
else
wait;
end if;
if END_SIM = FALSE then
TMP_clk <= '1';
wait for 50 ns;
else
wait;
end if;
end process;
ASSIGN_STIM_clk: STIM_clk <= TMP_clk;
ASSIGN_STIM_rst: STIM_rst <=
WPL.SIGNALS(TEST_PINS'pos(rst)+1);
ASSIGN_STIM_start: STIM_start <=
WPL.SIGNALS(TEST_PINS'pos(start)+1);
UUT: topmodule
port map( => ,
clk => STIM_clk,
rst => STIM_rst,
start => STIM_start,
=> ,
=> ,
read1 => STIM_read1,
=> );
end TB_ARCHITECTURE;
end TESTBENCH_FOR_topmodule;
```

VII. RESULTS

A. Simulation Results

For the evaluation of the suggested design methodology an analog signal is taken and processed, the observations obtained are as illustrated in below Fig. 7.

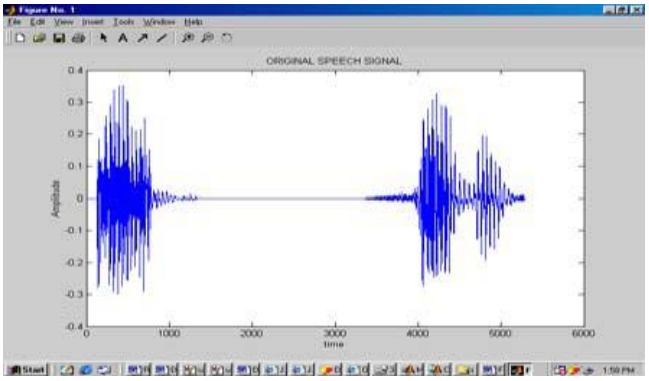

Fig. 7: Test signal sample for observation2

Samples Considered

0---------- - "0000000000000000"
-3.0518e-05--------"1000110000110101"
-3.0518e-05-------"1000110000110101"
0---------- - "0000000000000000"
-6.1035e-05-------"1000010000110110"
0---------- - "0000000000000000"
-6.1035e-05-------"1000010000110110"
-7.1553e-05-------"1000011001001111"
-6.1035e-05-------"1000010000110110"
-6.1035e-05-------"1000010000110110"
-7.1553e-05-------"1000011001001111"
-7.1553e-05-------"1000011001001111"
-7.1553e-05-------"1000011001001111"
0---------- - "0000000000000000"
-7.1553e-05-------"1000011001001111"
-6.1035e-05------"1000010000110110"
-7.1553e-05-------"1000011001001111"
-3.0518e-05------"1000110000110101"
-6.1035e-05-------"1000010000110110"
-3.0518e-05------"1000110000110101"
-6.1035e-05-------"1000010000110110"
0---------- "00000000000000000"

The above floating point numbers corresponding plot can be shown in Fig. 8.

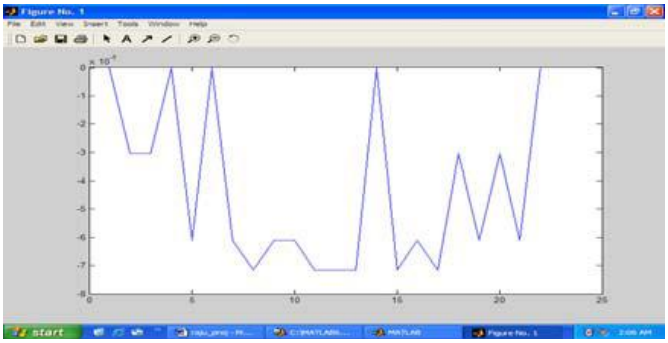

Fig. 8: Plot for the considered input signal sample or the samples sent at the input of the decomposer module

B. Simulation of proposed design Using ModelSimP.E10.1b

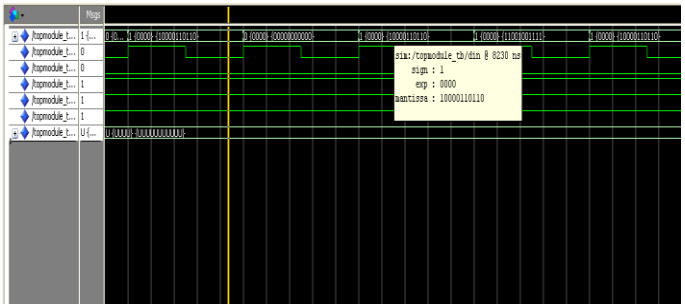

Fig. 9: Simulation of input sample float value

The above simulation result shows for -6.1035e-05. It's equivalent floating point number is 1 0000 10000110110. It is one of the floating point sample values from table.1. It has the sign bit: 1, exponential: 0000 and mantissa: 10000110110. For simplicity we have shown only one sample value. It has been sent as input at the input of Fig. 5. The same output got at the output of Fig. 5. In Fig. 9 the last value of the first row shows that 1 0000 10000110110 which is the input of decomposer shown in Fig. 5.

The yellow box of Fig. 9 shows that sign:1,exp:0000,mantissa: 10000110110 which is the output of reconstructor shown in Fig. 6. Hence, it is observed that both input and outputs are same.

C. Comparison of the input and output results

Fig. 10 shows the output of the reconstructer module for all sample values.

If we observe Fig. 8 and Fig. 10, the samples sent at the input of the decomposer module and the samples received at the output of the reconstructer module are same**.**

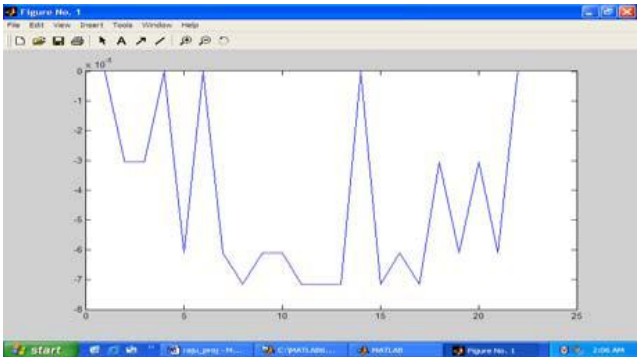

Fig. 10: The samples received at the output of the reconstructer module

D. Synthesis of proposed design using Xilinx-ISE13.4

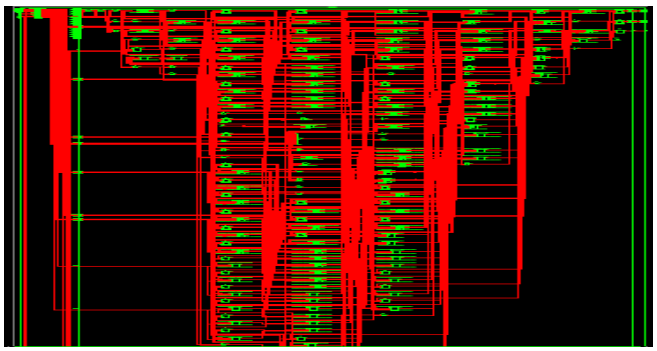

Fig. 11: Complete internal architecture of DWT

Fig. 11 shows the complete internal architecture of DWT. It has been synthesized using Xilinx-ISE13.4

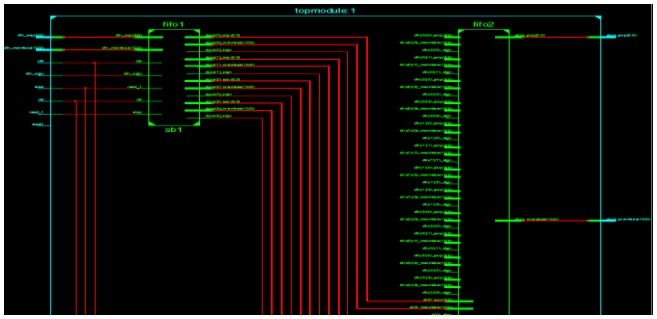

Fig. 12: Internal architecture between fifo1 and fifo2

Fig. 12 shows that internal architecture between fifo1 and fifo2. For simplicity we are showing only some of the synthesized diagrams.

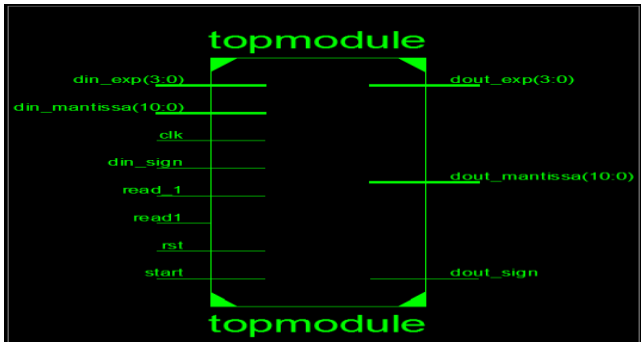

Fig. 13: Top module of DWT

Fig. 13 shows the Top module of DWT. It consists of different inputs and outputs of the Discrete Wavelet Transform.

VIII. CONCLUSION

In this work, we have added the analog features in the form of floating point notation, defined in the working library to the resource library of the ModelSim PE 10.1b tool successfully. Using this novel tool, analog mixed signal design i.e., DWT has been simulated. It has been used IEEE 754 standard based floating point representation. The design algorithms have been coded in VHDL [7]. Actually, the ModelSim PE 10.1b tool is defined for the digital design simulation. But, as we have provided the analog feature feasibility in the form of floating point arithmetic, the ModelSim PE 10.1b tool has been customized; hence the single tool could be used for analog-mixed signal designs.

REFERENCES

[1] D. Goldberg, "What every computer scientist should know about floating-point arithmetic" pp. 5-48 in ACM Computing Surveys vol. 23-1 (1991).

[2] Charles Farnum, "Compiler Support for Floating-Point Computation" Software Practices and Experience, pp. 701-9 vol. 18, July 1988.

[3] M. Leeser, X. Wang, " Variable Precision Floating Point Division and Square Root", Department of Electrical and Computer Engineering Northeastern University.

[4] Taek-Jun Kwon, Jeff Sondeen, Jeff Draper USC Information Sciences Institute Design Trade-Offs institute "Floating-Point Unit Implementation for Embedded and Processing-In-Memory Systems" 4676 Admiralty Way Marina del Rey, CA 90292 U.S.A.

[5] IEEE computer society: IEEE Standard 754 for Binary Floating-Point Arithmetic, 1985.

[6] Hierarchical VHDL Libraries for DSP ASIC Design, *John* McCanny', Douglas Ridge, Yi Hu, Jill Hunter, 1997 IEEE,

[7] J.Bhaskar,"AVHDL primer" Pearson education,2004.

[8] C. Burus, et al. Introduction to Wavelets and Wavelet Transformation A Primer. Prentice Hall, 1998.

[9] ASIC Design Methodology with On-Demand Library Generation, *Hidetoshi Onodera, Masanori Hashimoto, and Tetsutaro Hashimoto* Department of Communications and Computer Engineering, Kyoto University, 2001 Symposium on VLSl Circuits Digest of Technical Papers

[10] Development of User-defined Block Library for Active-Disturbance-Rejection-Control, *Jiang Ping, Bingshu Wang*, 2010 IEEE
[11] Modelsim PE SE 10.1b Performance Guidelines, Model Technology, December 3rd, 2010, User's manual,version 10.1b, Mentor Graphics Corporation.
[12] Fundamentals of digital logic with vhdl design 2e Stephen Brown, Zvonko Vranesic.pdf

R. Prakash Rao, received his M.Tech degree from College of Engineering Andhra University, Vizag, India and B.Tech degree from Siddardha College of Engineering, Nagarjuna University, Guntur, India. Presently he is working as Professor and HOD in St. Peter's Engineering College, Hyderabad. He published 08 papers in various National and International Journals and Conferences. He got best faculty award during 2009-2010 in ASTRA, Hyderabad. He has guided 02 M.Tech projects and about 25 B.Tech projects in various levels.

Dr. B.K. Madhavi, received Ph.D from JNTU, Hyderabad. She completed ME from BITS-PILANI in the specialization of Microelectronics. She published 24 research papers in various National and International Journals and Conferences. Presently she is guiding 10 PhD Students and guided several BTech and MTech Projects. She is also being reviewed research papers for IETE. She participated in several workshops, summer and winter schools, National, International conferences and also organized several National level workshops, student paper contests, and seminars etc.