IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 6, Issue 3, Ver. II (May. -Jun. 2016), PP 12-17
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Design and Development of Library Packages for Mixed-Signal Designs

R.Prakash Rao[1], Dr.B.K.Madhavi[2], P.Vijaya Bhaskar Rao[3]
[1]*Assoc.Professor, Department of ECE, St.Peter's Engineering College, JNTUH, India*
[2]*Professor, Department of ECE, Sridevi Women's Engineering College, JNTUH, India*
[3]*Asst.Professor, Department of ECE, St.Peter's Engineering College, JNTUH, India*

Abstract: *Analog to Digital Converters (ADCs) & Digital to Analog Converters (DACs) are called mixed-signal designs. ADC and DAC are also called CODER and DECODER respectively or the CODEC. ADC and DAC are used either for data processing applications or signal processing applications. To design analog to digital converters of data processing applications the normal sampling rate is sufficient whereas for signal processing applications over sampling is necessary since the signal is to be reconstructed well at the output of the receiver. But, the over sample ADC like Sigma-Delta, is limited with the speed. It can give the speed at the maximum of 1Mhz. As it can be observed, even the co-processors of DSP like audio CODEC operate with the speed of hundreds of Mhz. Hence, to eliminate the speed limitation of the oversampled ADC, floating point library packages for adder, multiplier are designed along with shifter to form the floating point Multiplier Accumulator Content (MAC) and will be compiled with the standard library packages of IEEE of any digitally defined simulation tool like Modelsim. Using such newly upgraded mixed signal simulation tool, mixed-signal Integrated Circuits like audio CODECs could be designed.*

Keywords: *ADC; DAC; CODEC; Sigma-Delta; MAC; Mixed signal designs; Simulation tool.*

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. 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 inter process 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. In order to eliminate the above said difficulties to design the mixed signal designs, a new approach is proposed using which mixed-signal designs are designed with customized simulation tool by single design flow.

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[1]. Such type of mixed-signal circuits are developed by designing the 32-bit floating point adder, floating point multiplier along with shifter to form the Multiplier Accumulator Content(MAC). MAC is the basic building block of any DSP system. Hence, the floating point MAC which is designed will be compiled with the standard library packages of IEEE of any digitally defined simulation tool like Modelsim. Using such newly upgraded mixed signal simulation tool, mixed-signal Integrated Circuits could be designed. Since, the mixed-signal systems are

designed using 32-bit floating point MAC, the input analog signal is decomposed into IEEE 754 single precision format with 32 bit length as shown in below figure(1).The single precision IEEE format consists of 1-bit is the sign bit, next 8-bits are exponent and the immediate followed 23-bits are the mantissa.

Sign(1)	Exponent (8)	Mantissa (23)

Figure (1): IEEE 754 single precision format

III. Floating Point Arithmetic

In most of the cases, the performance of the DSP system depends on the type of arithmetic is used. Basically, the arithmetic operations are three types in the DSP systems. a) fixed point arithmetic, b) floating point arithmetic c) block floating point arithmetic. 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 representation - the most common solution - basically represents reals in scientific notation. Scientific notation represents numbers as a base number and an exponent. In block floating point, common exponents are brought in to one bracket and within the bracket, again fixed point operation is performed. The floating point values are represented using Institute of Electrical and Electronics Engineers (IEEE) 754 standard. It can be briefly dealt as given below.

III.I IEEE 754 Floating Point Standard:

IEEE 754 floating point standard is the most common representation today for real numbers on computers. The IEEE has produced a Standard to define floating-point representation and arithmetic[2]. The standard brought out by the IEEE come to be known as IEEE 754[5]. 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.

III.I.I Floating point adder:

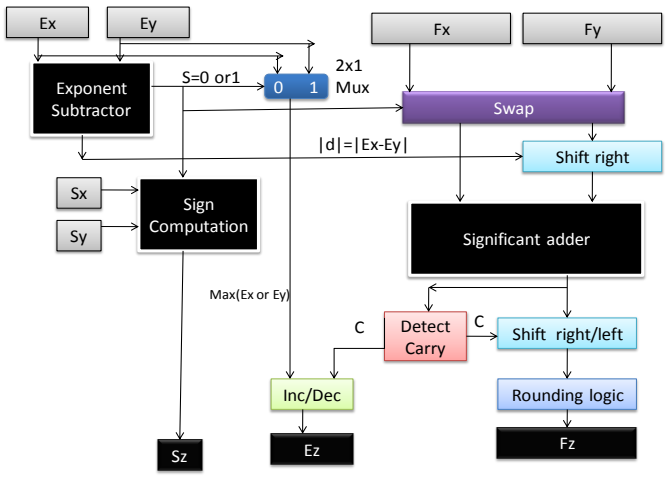


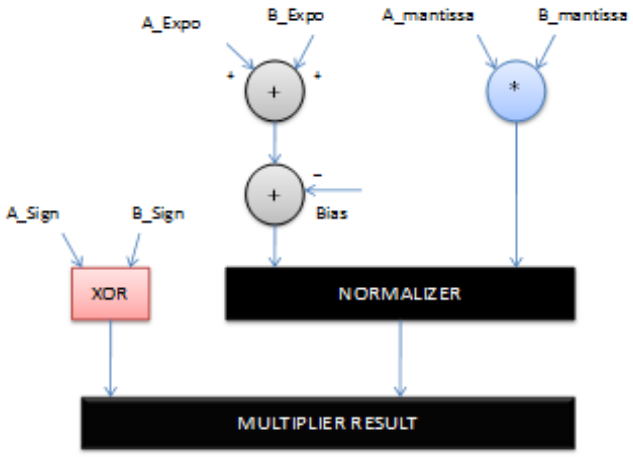
Figure (2): floating point adder

The basic description of the floating-point adder shown in figure(2) is illustrated as given below.

- If any of the input operand is a special value, the result is known before hand, thus all the computation can be bypassed.
- Mantissas are added or subtracted depending on the effective operation:
  - $S = (mx \pm (my \times 2ey - ex) \times 2ex$ if ex _ ey,
  - $((mx \times 2ex - ey) \pm my) \times 2ey$ if ex < ey.

- In order for the addition or subtraction to take place the binary point must be aligned. To achieve this, the mantissa of the smaller operand is multiplied by 2 difference of the exponents . This process is called alignment.
- The exponent of the result is the maximum of ex and ey; ez = max(ex, ey).
- If eop is addition, a carry-out can be generated and if eop is subtraction, cancellation might Occur. In each case normalization is required and consequently the exponent ez is updated.
- The exact result S is rounded to fit in the target precision. Sometimes rounding causes an overflow of the mantissa; a post-normalization is required.
- Determine exceptions by verifying the exponent of the result.

III.I.II Floating Point Multiplier:



Figure(3) : floating point multiplier

The basic description of floating-point multiplier shown in figure (3) is illustrated as given below[3].

- Verification of special values: if any of the input operand is a special value, the result is known beforehand and thus no computation is required.
- Mantissas are multiplied to compute the product asP = mx × my.
- The exponent of the result is computed as ez = ex + ey − 127. In biased representation, the bias 127 (for single precision) has to be subtracted.
- The sign of the result is computed as sz = sx XOR sy.
- The product might be unnormalized. In this case, normalization is required and consequently the exponent ez is updated.
- The exact result P is rounded according to the specified mode to produce the mantissa of the result mz. In case of post-normalization, it is necessary to increment the result exponent as ez = ez + 1.
- Determine exceptions by verifying the exponent of the result. When overflow occurs, the result is ±1 and when underflow occurs, the result is zero or a subnormal number (if the gradual underflow is supported in the implementation).

III.I.III Multiplier Accumulator Content(MAC):

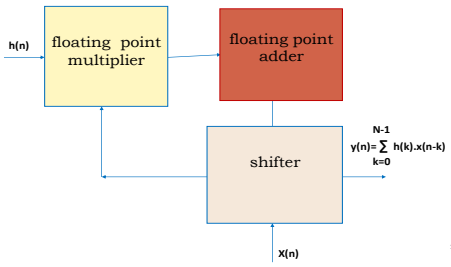


Figure (4): Floating Point MAC

The above figure(4) shows the multiplier accumulator content(MAC). Its operation can be illustrated as given below[4].

- Multiply the data sample x(n-k), with the co-efficient h(k).
- Add the current product to the previous output.

IV. Development Of Library Package

A library package is defined with user defined record data type as :

type real_single is
record
sign : std_logic;
exp: std_logic_vector(7 downto 0);
mantissa: std_logic_vector(22 downto 0);
end record;

The floating point notation is implemented using 32 bit IEEE-754 standard as presented below.

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 audio CODEC to design the high pass and low pass filters, db4 filter co-efficients are used because, db4 co-efficients are extracted for the meant for audio applications. Similarly, db6 meant for EEG and ECG applications to estimate the Tumors by C.T scan[5].

V. Update The Modelsim Tool

Since, the ModelSim is the user friendly tool, the VHDL packages designed for floating point adder , multiplier along with shifter are compiled with standard IEEE packages of the tool. Compilation of floating point arithmetic operators are shown step by step as given below.

1. 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[6].

2. 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 research into it. Start by creating a new directory for this exercise.

ii) Start ModelSim if necessary.

- 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, then Click **OK**.

When you pressed OK in above step, several lines were printed to the Main window Transcript pane:

vlib work
vmap 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.

VI. Results

A) Floating Point Adder:

The following snapshot shown in fig 6.1 had been taken from upgraded ModelSim after the timing simulation of the floating point Adder.

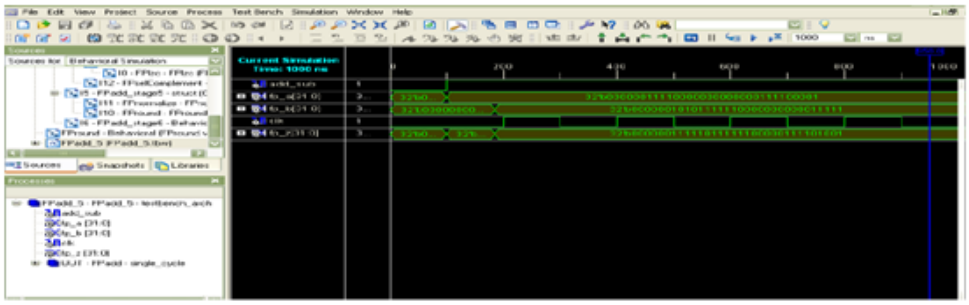


Figure 6.1: Output of Single Precision Floating Point Adder when above input's were given

B) Floating Point Multiplier:

The following snapshot shown in fig 6.2 had been taken from upgraded ModelSim after the timing simulation of the floating point multiplier.

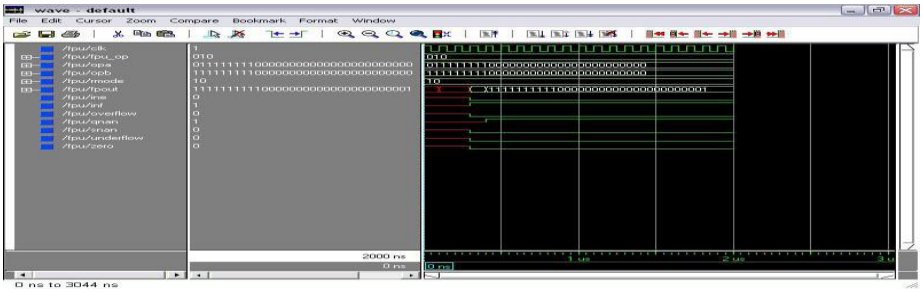


Figure 6.2: Output of Single Precision Floating Point Multiplier when above input's were given

c) Floating Point MAC:

The following snapshot shown in fig 6.3 had been taken from upgraded ModelSim after the timing simulation of the floating point MAC.

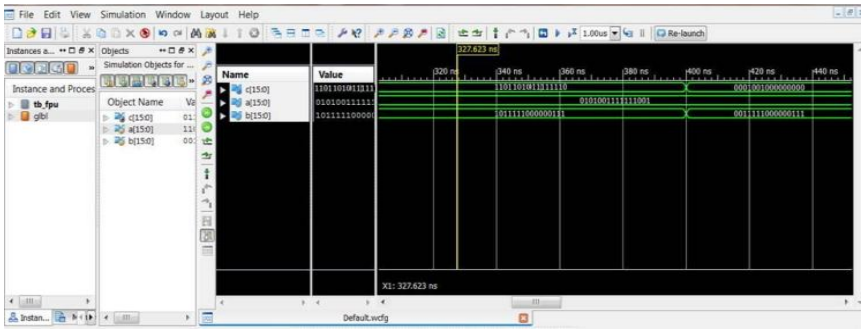


Figure 6.3: Output of Single Precision Floating Point MAC when above input's were given

VII. Conclusion

The main concentration of this research is to develop the library packages for the mixed signal designs of DSP applications like ADCs and DACs using floating point approach. Here, the floating point packages are developed using floating point adder, multiplier along with shifter. These packages are compiled with the Modelsim 10.3c tool. This upgraded tool can be used to design the co-processor of any of the DSP applications like in audio applications and medical applications.

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]. Raguveer M.Rao, Ajit S. Bopatikar "Wavelet transforms introduction to theory and application, Addison-weswley, 2001.