Digital Hearing Aids Specific µDSP Chip Design by Verilog HDL

Soon-Suck Jarng*, Lingfen Chen*, You-Jung Kwon*

* Department of Information Control & Instrumentation, Chosun University, Gwang-Ju, Korea
(Tel : +82-62-230-7107; E-mail: ssjarng@chosun.ac.kr)

Abstract: The hearing aid chip described in this paper is an analog & digital mixed system. The design focuses on the µDSP core. This µDSP core includes internal time delays to two inputs from front and rear microphones. The paper consists of two parts; one is the composure and signal processing algorithm of digital hearing aids and the other is Verilog HDL codes for µDSP cores. All digital modules in the design were coded and synthesized by Verilog HDL codes which were verified by Mentor Graphics and Synopsis semiconductor chip design tools.

Keywords: Digital Hearing Aid, Verilog HDL, µDSP, Chip Design

1. INTRODUCTION

Digital hearing aids (HA) are composed with microphones, receiver, volume control, interface socket, push button, telecoil as well as digital amplifier. This paper describes the chip design aspects, signal processing algorithms and µDSP cores. Some of recent sophisticated digital HAs are introduced into HA markets. The HA market requires the smaller size such as 5mm x 3mm x 2 mm (Completely In the ear Canal), the smaller power consumption and the better sound quality. Before 2000, analog HAs are better than digital HAs in power consumption aspect, but since 2000 this was reversed.

$$Power = V \cdot I = 0.5 \cdot C \cdot v^2 \cdot f \quad (1)$$

where V and I are HA battery supplied voltage (1.3V) and current (<1mA), and C and v are capacitance and voltage on transistor. f is sampling frequency.

The digital amplifier is named as its amplifying and other advanced functions are adjustable by external interfacing. The main parts of the digital amplifier chip are A/D, µDSP, D/A, EEPROM, voltage regulator, clock generator. Roughly A/D and D/A spend about 25% (95 dB dynamic range) and 5% (85 dB dynamic range) of total power consumption while µDSP consume about 75% of total power. Therefore the improvement of HA µDSP is strongly required. One way of power consuming reduction may be turning over function between amplification and idle state when voice is not coming in.

Digital HA chip is manufactured by layering and packaging of individual micro-sized dies (analog dies: A/D, D/A, voltage regulator, clock generator, digital dies: µDSP, EEPROM). The CMOS technology is applied for the chip manufacturing and the common gate length is 0.25 µm but is now shortened down to 0.18 µm or 0.13 µm.

Sampling frequency is usually 16~32 kHz and clock cycles per one audio sample are normally 64~128. Also digital HA chips have 32 IIR Biquad filters and about 160 taps FIRs. The following is the performance of µDSP: Dual MAC and Dual ALU, 20 bits fixed point, 2.048 MHz clock speed, 2 MIPS, 4 M MACs, 0.48 mW power consumption (400 µA x 1.2V).

This paper shows only some parts of researched works on digital HA chip design.

2. DIGITAL HA CHIP SPECIFICATION

The Fig 1 shows the diagram of the hearing aids DSP Chip.

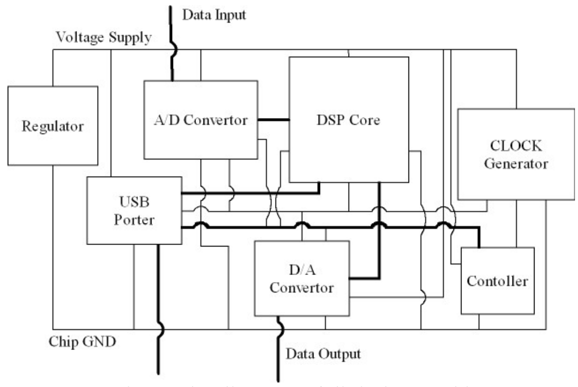

Fig. 1 The diagram of digital HA Chip

The regulator makes some unsteady supply voltage, which is from 1.2~1.3V battery, into a steady one. The clock generator generates a system clock signal. The controller produces control signals. When the power is switched on, the controller firstly checks the initial state of the clock pulse. The controller will give a start pulse immediately after the clock becomes steady. Also when a memory push button is clicked, the controller recognizes a new memory state and will initiate an interrupt signal to µDSP core, that is, the controller changes the content of the program counter and reinitiate the µDSP core to begin at the new memory state. We consider four memory states, so that the HA user may select one of four chip parameter settings; quiet room, office, market, outdoor environment.

The A/D and D/A converters convert signals between analog and digital systems. Two separate A/D and D/A converters are built in, so that the HA chip can have directivity function. Each of two A/D converters output digital signals to the µDSP core and the µDSP does time delays between two channels. The amount of the time delay is controlled by the user. The USB port receives the serially transferred digital signals from PC. The digital signals are divided into two groups; instructions and parameters which are used for the adjusting of the HA performance. The USB port distributes the incoming serial data into parallel data and store them into

EEPROM, and vice versa. The programmed instructions are a set of operations to perform the desired signal processing algorithm. The parameter data are used for the algorithm operation.

The µ DSP core is the main module of the algorithm block in the HA DSP chip (Fig. 2).

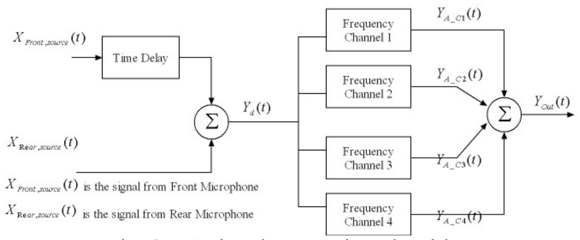

Fig. 2 HA signal processing algorithm

The µ DSP core adds internal time delay to two input signals, $x_{Front\_source}(t)$ and $x_{rear\_source}(t)$, which are from front and rear microphones respectively. The time delay determines the directivity of the HAs. We get the output of the adder, $Y_d(t)$. Then the output signal, $Y_d(t)$, is sent into four sub-channels. Each channel consists of a lowpass / bandpass / highpass filter and an amplifier. Fig 3 shows the structure of a frequency channel.

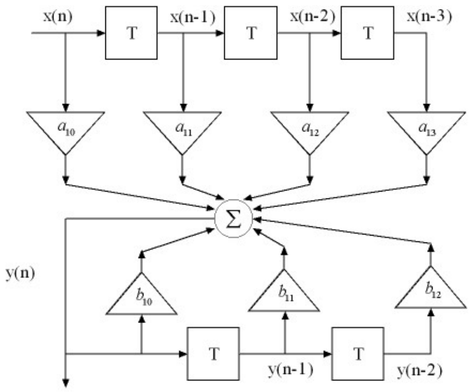

Fig. 3 IIR digital filter structure of a frequency channel

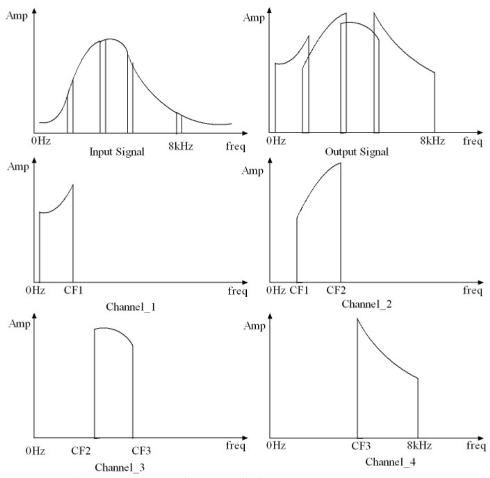

Fig. 4 Four channel frequency responses

In each channel, the signal is filtered with an IIR active filter of which coefficients are determined by the fitting formula according to the hearing loss threshold of hearing impairments.

$$y_{CH1}(n)=a_{10}\cdot x(n)+a_{11}\cdot x(n-1)+a_{12}\cdot x(n-2)+a_{13}\cdot x(n-3)+b_{10}\cdot y(n-1)+b_{11}\cdot y(n-2)$$
$$y_{CH2}(n)=a_{20}\cdot x(n)+a_{21}\cdot x(n-1)+a_{22}\cdot x(n-2)+a_{22}\cdot x(n-3)+b_{20}\cdot y(n-1)+b_{21}\cdot y(n-2)$$
$$y_{CH3}(n)=a_{30}\cdot x(n)+a_{31}\cdot x(n-1)+a_{32}\cdot x(n-2)+a_{32}\cdot x(n-3)+b_{30}\cdot y(n-1)+b_{31}\cdot y(n-2)$$
$$y_{CH4}(n)=a_{40}\cdot x(n)+a_{41}\cdot x(n-1)+a_{42}\cdot x(n-2)+a_{42}\cdot x(n-3)+b_{40}\cdot y(n-1)+b_{41}\cdot y(n-2) \quad (2)$$

The last step of Fig. 2 is to add the processed signal from four sub-channels together.

$$y(n) = y_{CH1}(n) + y_{CH2}(n) + y_{CH3}(n) + y_{CH4}(n) \quad (3)$$

3. µ DSP Structure

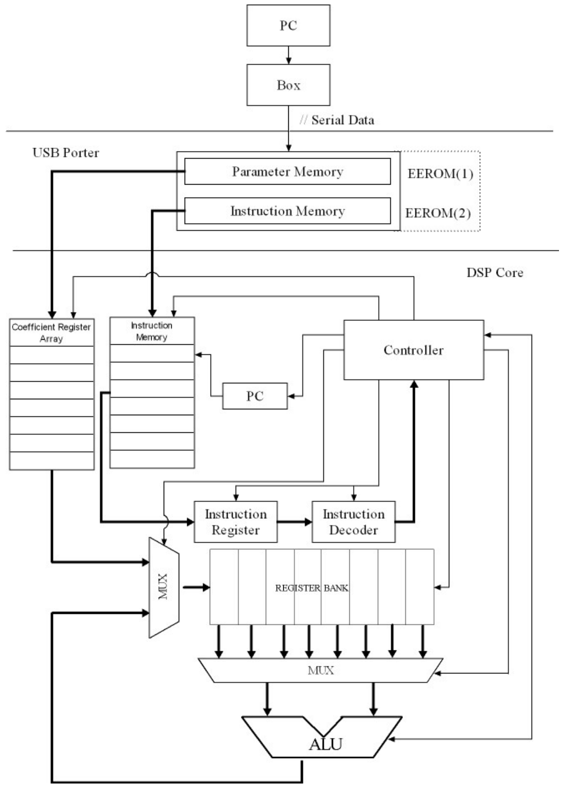

Fig 5 The Data Flow of The HA µ DSP

The chip works on three working modes, system initialization mode, memory selection mode, and signal processing mode.

(1) In the system initialization mode, the chip fetches the parameters and the instructions stored in USB EEPROM into core RAM memory. The resulted fitting instructions and parameters are already transferred into USB EEPROM. Whenever the supply power is turned on, the chip starts the system initialization mode.

(2) In the memory selection mode, the chip changes its processing mode. There are several (four) scene modes on system to manually adjust the HA to fit the acoustic circumstance. By clicking a memory push button, the user can rotary to different memory selection mode.

(3) In the signal processing mode, the chip one by one executes the instructions invoked to the core RAM memory in synchronization with the system clock.

Though both the USB interface and the µ DSP core belong to the same chip, they never work together. In the system initialization mode, if the user interrupts the system, the USB port communicates with the external device such as PC while the µ DSP operates independently from the USB port. Similarly when the memory selecting push button is clicked, the µ DSP stops its operation and fetches newly selected parameters and instructions from EEPROM in USB port. After that, the operations are the same as in the system initialization mode.

4. µ DSP Data Format

The Input signal voltage level from the microphone to the chip is between $\pm 10mV$ and $\pm 1.3V$. It determines the minimum unit of the signal level as 10 mV. The output voltage level should be less than the supply voltage 1.3 V. So we used a 16-bit fixed point 2's complementary binary numbers. The most significant bit is a sign bit. The zero represents a positive datum and one represents a negative datum. The second bit represents integer fraction while the rest 14 bits are decimal fraction.

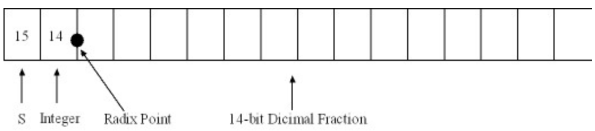

Fig. 6 16-bit fixed point 2's complementary binary numbers

Table 1. Binary Value Range

	S	Integer	Decimal Fraction	V
Positive Maximal	0	1	11111111111111	B(1.11111111111111)
Positive Minimal	0	0	00000000000001	B(0.00000000000001)
Negative Minimal	1	1	11111111111111	B(-0.00000000000001)
Negative Maximal	1	0	00000000000001	B(-1.11111111111111)

Positive Range: 2^{-14} (=6.103515625e-5) <= value < $2-2^{-14}$ (=1.99993896484375);

Negative Range: $-2+2^{-14}$ (=-1.99993896484375) <= value < -2^{-14} (=-6.103515625e-5);

The logic one is equivalent to about 61 µ V . The noise margin of the microphone output is about 50 µ V. Therefore 2^{-14} is quite good enough for the digital resolution.

5. ASM of µ DSP

Fig 7 shows the block diagram of the HA µ DSP Core. It consists of three blocks (Verilog HDL):

1. Control Unit (control.v) : The control unit is the main unit of the core. It plays roles of decoding instructions and controlling ALU and register banks according to decoded micro operations. It also serves the interrupts and works with BIU to handle instruction/parameter fetches.

2. ALU (alu.v) : The ALU for the HA µ DSP core is much simplified. It contains adder, multiplies, comparison, increment, decrement, and logic operation units.

3. Registers Bank and Bus Interface Unit (regu_biu.v) : The register bank contains 8 general purpose registers, stack pointer, program counter (PC), and the processor status word register. The general purpose registers are implemented as an 8 bits memory, so that it can be easily replaced by a 2-read/1-write per clock synchronous memory to reduce the area.

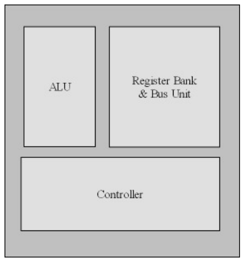

Fig 7 The block diagram of the HA µ DSP Core

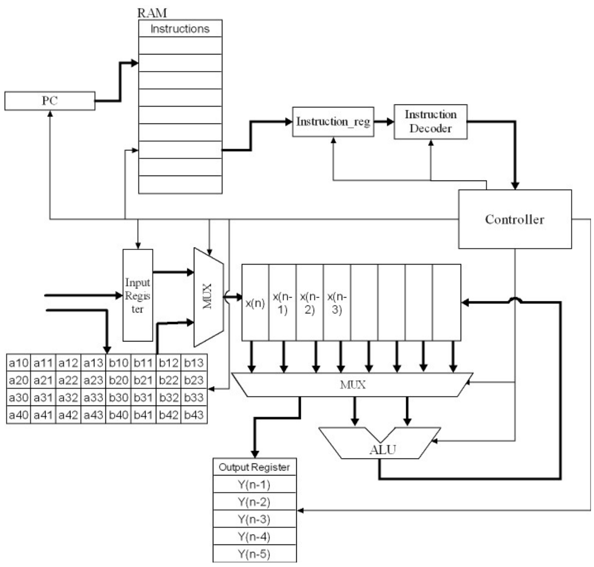

Fig 8 The internal structure of the HA µ DSP Core

Fig. 8 shows the internal structure of the HA µ DSP Core and Fig 9 shows the ASM(Algorithmic State machine) of the HA µ DSP Core.

6. Verilog Codes for µ DSP ALU

Compared to the standard CPU ALU, the ALU used in the HA µ DSP Core is quite simple. Because the operation types are no more than 10, we use a 4-bit operation code in a 16-bit instruction. The other 12-bits point to the register in which the operands are stored. Table 2 shows the operations of the ALU.

data_a – data_b = data_a + (–data_b) (4)

Table 2 ALU Operation Types

Operation		OPCODE	Description
Arithmetic	Add	0000	Rn1 + Rn2
	Sub	0001	Rn1 – Rn2
	Mult	0010	Rn1 * Rn2
	Incr	0011	Rn + 1
	Dcre	0100	Rn – 1
	Comp	0101	If Rn1 > Rn2, F=1 Else F=0
Logic	And	0110	Rn1 & Rn2
	Or	0111	Rn1 | Rn2
	Not	1000	~Rn
Move	Move	1001	Rn1 <= Rn2

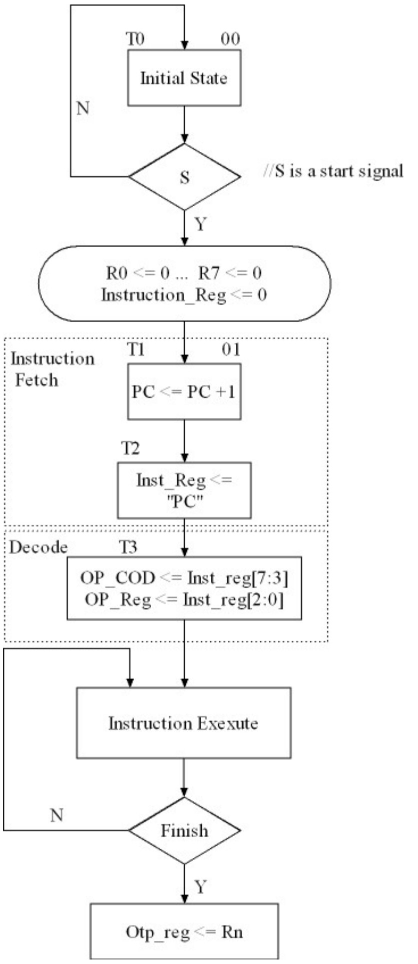

Fig 9 The ASM of the HA µ DSP Core

The following is some part of ALU Verilog HDL codes:

```
module   alu_11(data_a, data_b, op, data_out, cout, done );
    input [3:0] data_a, data_b;
    input [3:0] op;
    output [3:0]   data_out;
    output done, cout;
    parameter
        ALU_ADD        = 1'h0,
        ALU_SUB        = 1'h1,
        ALU_INC        = 1'h2,
        ALU_DEC        = 1'h3,
        ALU_MUL        = 1'h4,
        ALU_AND        = 1'h5,
        ALU_OR     = 1'h6,
        ALU_NOT        = 1'h7,
        ALU_MOV        = 1'h8;
reg [3:0] data_out;
reg [3:0] data_out_tl, data_out_th;
reg [3:0] temp_a, temp_b, temp;
reg done,cout;

always @( op or data_a or data_b )
begin
done = 1'b0 ;  // finish signal
cout = 1'b0 ;      // Overflow flag
data_out_tl = 4'b0000;
data_out_th = 4'b0000;
temp_a = data_a;
temp_b = data_b;

   case ( op )
        ALU_ADD :    // Suggestion: make a subprogram.
         begin
    adder(data_a, data_b, data_out_th, cout);
         end

        ALU_SUB :
         begin
    if ( temp_b == 4'b0000    )          // The temp_b is zero
            ;
            else //    ---- 1
        begin
         if ( temp_b[3] )    // Nagetive changes to Positive
          begin
         temp = temp_b;
                temp[2:0] = ~(temp[2:0] - 1'b1);
         temp[3] = 0;
         temp_b = temp;
          end
         else              // Positive changes to Nagetive
          begin
         temp = temp_b;
                temp = (~temp[2:0]) + 1'b1;
         temp[3] = 1;
         temp_b = temp;
          end
     end // end of else ---- 1

    adder(temp_a, temp_b, data_out_th, cout);
         end
        ALU_INC :    data_out_th = data_a + 1;
        ALU_DEC :    data_out_th = data_a - 1;
        ALU_MUL :    {data_out_th,data_out_tl} = data_a * data_b;
        ALU_AND :    data_out_th = data_a & data_b;
        ALU_OR    :    data_out_th = data_a | data_b;
        ALU_NOT :    data_out_th = ~data_a;
        ALU_MOV :    data_out_th = data_a;
      default : data_out_th = data_out_th;
   endcase

 if ( cout == 0)              // Overflow Solusion
    begin
    data_out = data_out_th;
    done <= 1'b1;
    end
 else if((!data_a[3])&(!data_b[3])) // Positive
    begin
    data_out = 4'b0111;
```

```
        done = 1'b1;
        end
    else
        begin
          data_out = 4'b1000;
          done = 1'b1;
        end
  end

    task adder;
  input [3:0] data_a_st, data_b_st;
  output [3:0] data_out_st;
  output cout_st;
   begin
          if ( data_a_st[3]&data_b_st[3] )
        begin
            {cout_st, data_out_st} = data_a_st + data_b_st;
            if ( data_out_st[3] )
             cout_st =    1'b0;
      else
             cout_st =    1'b1;
      data_out_st[3] = 1'b1;
        end
      else if ( (!data_a_st[3])&(!data_b_st[3]) )
        begin
      {cout_st, data_out_st} = data_a_st + data_b_st;
      if ( data_out_st[3] )
          cout_st =    1'b1;
      else
          cout_st =    1'b0;
      data_out_st[3] = 1'b0;
        end
      else
        begin
      {cout_st, data_out_st} = data_a_st + data_b_st;
      cout_st = 1'b0;
        end
     end
  endtask
  endmodule
```

And the following is Verilog HDL codes for ALU addition test bench:

```
`timescale 1ns/1ns
`include "alu_11.v"

module    tb_alu_11add;
    reg [3:0] data_a, data_b;
    reg [3:0] op;
    wire [3:0]    data_out;
    wire done, cout;
    alu_11 u0(data_a, data_b, op, data_out, cout, done);
    initial
        begin
        data_a = 4'b0000;
        data_b = 4'b0000;
        op = 4'h0;
        repeat (256)
        begin
          if ( data_a == 4'b1111 )
            begin
                    data_a = 4'b0000;
              data_b = data_b + 1'b1;
        end
          else
              #10    data_a = data_a + 1'b1;
        end
       end
     initial
     $monitor("data_a = %b data_b = %b Overflow = %b data_out = %b", data_a, data_b, cout, data_out);
endmodule
```

And the following is Verilog HDL codes for ALU subtraction test bench:

```
`timescale 1ns/1ns
`include "alu_11.v"

module    tb_alu_11sub;
    reg [3:0] data_a, data_b;
    reg [3:0] op;
    wire [3:0]    data_out;
    wire done, cout;
    alu_11 u0(data_a, data_b, op, data_out, cout, done);
    initial
        begin
        data_a = 4'b0000;
        data_b = 4'b0000;
        op = 4'h1;
        repeat (256)
        begin
          if ( data_a == 4'b1111 )
            begin
                    data_a = 4'b0000;
              data_b = data_b + 1'b1;
        end
          else
              #10    data_a = data_a + 1'b1;
        end
       end
     initial
     $monitor("data_a = %b data_b = %b Overflow = %b data_out = %b", data_a, data_b, cout, data_out);
    endmodule
```

7. Results of µ DSP ALU

The following is results for ALU addition test bench:

```
data_a = 0001 data_b = 0000 Overflow = 0 data_out = 0001
data_a = 0010 data_b = 0000 Overflow = 0 data_out = 0010
data_a = 0011 data_b = 0000 Overflow = 0 data_out = 0011
data_a = 0100 data_b = 0000 Overflow = 0 data_out = 0100
data_a = 0101 data_b = 0000 Overflow = 0 data_out = 0101
data_a = 0110 data_b = 0000 Overflow = 0 data_out = 0110
data_a = 0111 data_b = 0000 Overflow = 0 data_out = 0111
data_a = 1000 data_b = 0000 Overflow = 0 data_out = 1000
data_a = 1001 data_b = 0000 Overflow = 0 data_out = 1001
data_a = 1010 data_b = 0000 Overflow = 0 data_out = 1010
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
data_a = 0110 data_b = 1111 Overflow = 0 data_out = 0101
data_a = 0111 data_b = 1111 Overflow = 0 data_out = 0110
data_a = 1000 data_b = 1111 Overflow = 1 data_out = 1000
data_a = 1001 data_b = 1111 Overflow = 0 data_out = 1000
data_a = 1010 data_b = 1111 Overflow = 0 data_out = 1001
data_a = 1011 data_b = 1111 Overflow = 0 data_out = 1010
data_a = 1100 data_b = 1111 Overflow = 0 data_out = 1011
data_a = 1101 data_b = 1111 Overflow = 0 data_out = 1100
data_a = 1110 data_b = 1111 Overflow = 0 data_out = 1101
data_a = 0000 data_b = 0000 Overflow = 0 data_out = 0000
```

And the following is results for ALU subtraction test bench:

data_a = 0001 data_b = 0000 Overflow = 0 data_out = 1001
data_a = 0010 data_b = 0000 Overflow = 0 data_out = 1010
data_a = 0011 data_b = 0000 Overflow = 0 data_out = 1011
data_a = 0100 data_b = 0000 Overflow = 0 data_out = 1100
data_a = 0101 data_b = 0000 Overflow = 0 data_out = 1101
data_a = 0110 data_b = 0000 Overflow = 0 data_out = 1110
data_a = 0111 data_b = 0000 Overflow = 0 data_out = 1111
data_a = 1000 data_b = 0000 Overflow = 1 data_out = 1000
data_a = 1001 data_b = 0000 Overflow = 1 data_out = 1000
~~~~~~
data_a = 0110 data_b = 1111 Overflow = 0 data_out = 0111
data_a = 0111 data_b = 1111 Overflow = 1 data_out = 1000
data_a = 1000 data_b = 1111 Overflow = 0 data_out = 1001
data_a = 1001 data_b = 1111 Overflow = 0 data_out = 1010
data_a = 1010 data_b = 1111 Overflow = 0 data_out = 1011
data_a = 1011 data_b = 1111 Overflow = 0 data_out = 1100
data_a = 1100 data_b = 1111 Overflow = 0 data_out = 1101
data_a = 1101 data_b = 1111 Overflow = 0 data_out = 1110
data_a = 1110 data_b = 1111 Overflow = 0 data_out = 1111
data_a = 0000 data_b = 0000 Overflow = 0 data_out = 1000

8. Conclusion

This paper shows only some parts of researched works on digital HA chip design. There are lots of works to be done in order to completely finish the present study on HA.

ACKNOWLEDGMENTS

This study was supported by research fund from the ministry of commerce, industry and energy (MOCIE Korea) standardization technology development project (standardization study on digital hearing aid: project number 10016821) in 2004.

REFERENCES

[1] M.M.Mano, “Digital Design”, 3rd Edition, Prentice Hall, 2002.
[2] Harvey Dillon, "Hearing Aids", Printed by Thieme, 2001.
[3] http://www.gennum.com/hip/dproducts/paragon/gb3211.htm
[4] http://www.gennum.com/hip/dproducts/duet/gb3212.htm
[5] http://www.dspfactory.com/technology/signaklaratechnology.html
[6] C. Lee, “Recent digital signal processing”, published by BooksHill Co., 2002.
[7] Gennum Co., “Biquad filters in PARAGON digital hybrid”, Doc. No. 20205-1, 2001.
~~~~~~