Lowell Jacobson and Hanneul Earl Han

Structural Problems

The primary structural flaw in our project was the fact that our group consisted of only two students. The project itself was overwhelming for two students, especially with the fact that Earl was taking 7 classes this semester. With the amount of VHDL and C coding necessary to implement the ambitious aims of the design document, a group size of 3 to 4 students would have been more appropriate. A larger group would also allow for increased feedback and more collaboration among group members, as well as pointing out conceptual and coding errors when we got stuck. In the future, we believe that two member groups should be strongly discouraged for all but the strongest students.

In addition to the group size, there was not enough interaction between the TAs and the students. Often the TAs came into the lab at odd hours, when some students were unavailable. Other times, we would we could make it into the lab, particularly with Earl and his schedule, there were no TA's on hand, and we had to deal with the difficulties of explaining the problem via email.

The second half of the class should be structured so that there are set times when TAs are in the lab, not just at 11-12:30 most nights. In addition each group should have set meeting times with their particular TA each week, in which the TA sees the progress since last week, and makes suggestions if the students are going down the wrong path. The scheduled class meeting time would be ideal, especially because it's guaranteed (barring conferences) that the professor and the students will be free during that time. In addition, the TA should meet with the students of each group before the design document

is due (so the students have an idea of the limitations involved, data format, and the like), and after the design document is submitted, so that the students and the TA can figure out more specifically what needs to be done. Of particular concern were the difficulties with the OPB VHDL files, as we had a difficult time grasping exactly how they should be used. The TAs should work closely with the groups to make sure progress is being made, and that major unforeseen problems are dealt with in a timely fashion.

For students who are particularly weak in programming in C and VHDL, optional tutorial sessions can be created for students to attend, or in-class labs can be implemented so that the class as a whole has a common level of understanding, thus not splintering the class to experienced students and novices. It would be more helpful if somehow groups can be more balanced in terms of skill level. Not only were we a two-person group, but also both of us felt we were the weaker students in the class. We feel if our group had more experienced students, more work on the project could be completed more efficiently, and weaker member can learn and follow along.

Technical Problems

In the technical realm, the major problems arose in memory allocation in the VHDL. We realized too late that essentially the FPGA does not have enough memory to implement the delay elements for the signal processing in VHDL, and instead needed to both read and write to RAM in continuous loops. This however, would have taken much more planning and time, and by the time we realized that this was the reason for the errors, we did not have nearly the time to code a solution, let alone research and plan the implementation of reading and writing to the on-board RAM addresses..

Tone control implementation proved to be difficult because of the complex communication between the C code and the VHDL. Also, fades were originally going to be controlled within the codec itself, because the codec has registers that control various fade options. However, the TA (Christian) informed us that the manipulation of the registers directly in the codec is extremely difficult, and we should not even bother to try.

For the user interface, handling user input is a much more involved process than we realized, with the board having to constantly check to see input updates from the user, similar in concept to the typewriter lab. Due to time constraints, Earl planned to use stdlib library and the scanf function to handle user input. However the Xilinx board does not support scanf function as most non-embedded programmers understand it, as it lacks an underlying operating system to handle all of the interrupts. It seems the only way to truly implement user input is to complete the process discussed in the beginning of this paragraph, which is another part that we did not realize until too late. The VHDL files were essentially correct, however, the implementation was not, as they needed to be written to and read from RAM. If the FPGA were MUCH larger, than the implementation of the audio effects would have been successful.

The project itself was a valuable learning experience to have some familiarity with VHDL and Xilinx tools. Although the project did not function properly, we feel we learned much about implementation of VHDL programming, and especially about the level of detail and planning involved in such an undertaking. Perhaps the biggest lesson to us, as Prof. Edwards put it, is that “the squeaky wheel gets the oil,” and that we needed to be much more vocal and proactive in finding out the problems in our project and seeking constant assistance, especially with a group as small as ours. While neither of us

is going further in engineering, we have learned many lessons from this class and this project that will be applicable both to our future academic endeavors and to our careers.

Who did what

Lowell: decided on the project, wrote all the VHDL, wrote the proposal and the design document.

Earl: wrote the C interface.

Lowell and Earl together: wrote the Future Considerations/Final writeup.

VHDL Files Written:

```
-- Audio Effects Unit: Hierarchical

-- 1 to 4 DEMUX for beginning of audio effects unit
library ieee;
use ieee.std_logic_1164.all;
entity demux_1_to_4 is
  port (MUX_CONTROL: in std_logic_vector(1 downto 0); -- Control logic for
MUX/DEMUX
        PIN: in std_logic; --16 bit one-channel stereo audio sample
        S: out std_logic_vector(0 to 3));
end demux_1_to_4;

architecture function_table_ws of demux_1_to_4 is
begin
  with MUX_CONTROL select
    PIN  => S(0) when "00",         -- used to be I
         S(1) when "01",
         S(2) when "10",
         S(3) when "11",
         'X' when others;
   S0 <= S(0); -- Normal Operation
   S1 <= S(1); -- Flanging
   S2 <= S(2); -- Echo
   S3 <= S(3); -- Reverb/Vibrato
end function_table_ws;
-- COMPLETE 1 to 4 DEMUX


-- FLANGING UNIT ATTACHED to O(1) and E(1)

  -- 0 to 47 counter (~1ms) to act as a clock with period of ~ 1ms
  library ieee;
  use ieee.std_logic_1164.all;
  entity 1_ms_clock is
     port (AU_LRCK : in std_logic;      -- takes in fast clock
                MSC : out std_logic);    -- outputs clock with T = 1ms
  end 1_ms_clock;

  architecture behavioral of 1_ms_clock is
    begin
    count = 0;
    up <= 0;
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      if (count + 1)%98 = '0' then
        count = 0;
        up <= not up;
          else
            up <= 1;
          endif;
      endif;
    endif;
  end process;
    MSC <= up;
  end behavioral;

  -- 1 to 10 to 1 counter controlling select on 1_to_10 DEMUX, on 1 ms clock
  library ieee;
  use ieee.std_logic_1164.all;
  use IEEE.STD_LOGIC_ARITH.ALL;
  entity bouncer_1_to_10 is
    port (MSC : in std_logic; -- takes MSC (1 ms clock), not AU_BCLK
          BCR : out std_logic); -- bounce signal
  end bouncer_1_to_10;
```

```
architecture behaviorial of bouncer_1_to_10 is
  up <= 0;
  count <= 1;
begin
process (MSC)
begin
  if (MSC'event and (MSC = '1')) then
    if up = '0' then
      count = count + 1;
    else
      count = count - 1;
  endif;
  if count = '1' or count = '10' then
    up <= not up;
  endif;
end process;
  BCR <= count;
end behavioral;

-- ======================================================
--    FLANGER DELAY ELEMENTS
-- All of these use the 98 KHz clock

-- FOR ELEMENT N (N!=1):
--    input is TN
--    output is TNO
-- FOR ELEMENT 1
--    input is S1 from effect select DEMUX
-- ======================================================
  -- ~1 ms delay
  library ieee;
  use ieee.std_logic_1164.all;
  entity srg_1 is
    port(AU_LRCK, S1 : in std_logic;
    QT1  : out std_logic_vector(97 downto 0);
    T1O : out std_logic);
  end srg_1;

  architecture behavioral of srg_1 is
  signal shift : std_logic_vector(97 downto 0);
  begin
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      shift <= shift(96 downto 0) & S1;
    endif;
  end process;
    QT1 <= shift;
    T1O <= shift(97);
  end behavioral;

  -- 2 ms delay
  library ieee;
  use ieee.std_logic_1164.all;
  entity srg_2 is
    port(AU_LRCK, T1O : in std_logic;
    QT2  : out std_logic_vector(97 downto 0);
    T2O : out std_logic);
  end srg_2;

  architecture behavioral of srg_2 is
  signal shift : std_logic_vector(97 downto 0);
  begin
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      shift <= shift(96 downto 0) & T2;
    endif;
  end process;
    QT2 <= shift;
    T2O <= shift(97);
```

```
end behavioral;

-- 3 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_3 is
  port(CLK, T2O : in std_logic;
  QT3  : out std_logic_vector(97 downto 0);
  T3O : out std_logic);
end srg_3;

architecture behavioral of srg_3 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T3;
  endif;
end process;
  QT3 <= shift;
  T3O <= shift(97);
end behavioral;

-- 4 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_4 is
  port(AU_LRCK, T3O : in std_logic;
  QT4  : out std_logic_vector(97 downto 0);
  T4O : out std_logic);
end srg_4;

architecture behavioral of srg_4 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T4;
  endif;
end process;
  QT4 <= shift;
  T4O <= shift(97);
end behavioral;

-- 5 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_5 is
  port(AU_LRCK, T4O : in std_logic;
  QT5  : out std_logic_vector(97 downto 0);
  T5O : out std_logic);
end srg_5;

architecture behavioral of srg_5 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T5;
  endif;
end process;
  QT5 <= shift;
  T5O <= shift(97);
end behavioral;

-- 6 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_6 is
```

```
  port(AU_LRCK, T5O : in std_logic;
  QT6  : out std_logic_vector(97 downto 0);
  T6O : out std_logic);
end srg_6;

architecture behavioral of srg_6 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T6;
  endif;
end process;
  QT6 <= shift;
  T6O <= shift(97);
end behavioral;

-- 7 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_7 is
  port(AU_LRCK, T6O : in std_logic;
  QT7  : out std_logic_vector(97 downto 0);
  T7O : out std_logic);
end srg_7;

architecture behavioral of srg_7 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T7;
  endif;
end process;
  QT7 <= shift;
  T7O <= shift(97);
end behavioral;

-- 8 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_8 is
  port(AU_LRCK, T7O : in std_logic;
  QT8  : out std_logic_vector(97 downto 0);
  T8O : out std_logic);
end srg_8;

architecture behavioral of srg_8 is
signal shift : std_logic_vector(97 downto 0);
begin
process (AU_LRCK)
begin
  if (AU_LRCK'event and (AU_LRCK = '1')) then
    shift <= shift(96 downto 0) & T8;
  endif;
end process;
  QT8 <= shift;
  T8O <= shift(97);
end behavioral;

-- 9 ms delay
library ieee;
use ieee.std_logic_1164.all;
entity srg_9 is
  port(AU_LRCK, T8O : in std_logic;
  QT9  : out std_logic_vector(97 downto 0);
  T9O : out std_logic);
end srg_9;

architecture behavioral of srg_9 is
```

```
    signal shift : std_logic_vector(97 downto 0);
    begin
    process (AU_LRCK)
    begin
      if (AU_LRCK'event and (AU_LRCK = '1')) then
        shift <= shift(96 downto 0) & T9;
      endif;
    end process;
      QT9 <= shift;
      T9O <= shift(97);
    end behavioral;

    -- 10 ms delay
    library ieee;
    use ieee.std_logic_1164.all;
    entity srg_10 is
      port(AU_LRCK, T9O : in std_logic;
      QT10  : out std_logic_vector(97 downto 0);
      T10O : out std_logic);
    end srg_10;

    architecture behavioral of srg_10 is
    signal shift : std_logic_vector(97 downto 0);
    begin
    process (AU_LRCK)
    begin
      if (AU_LRCK'event and (AU_LRCK = '1')) then
        shift <= shift(96 downto 0) & T10;
      endif;
    end process;
      QT10 <= shift;
      T10O <= shift(97);
    end behavioral;


    -- ========================================
    -- FLANGER MULTIPLEXER: selects appropriately delayed data based on bouncing
counter
    -- ========================================
    library ieee;
    use ieee.std_logic_1164.all;
    entity mux_10_to_1 is
      port (BCR: in std_logic_vector(3 downto 0); -- Control logic for flanger
MUX from Bouncer
            D: in std_logic_vector(1 to 10);
            FOUT: out std_logic;)
      D(1) <= T1O;
      D(2) <= T2O;
      D(3) <= T3O;
      D(4) <= T4O;
      D(5) <= T5O;
      D(6) <= T6O;
      D(7) <= T7O;
      D(8) <= T8O;
      D(9) <= T9O;
      D(10) <= T10O;
    end mux_10_to_1;

    architecture function_table_ws of mux_10_to_1 is
    begin
      with BCR select
       FOUT <= D(1) when "0001",
               D(2) when "0010",
               D(3) when "0011",
               D(4) when "0100",
               D(5) when "0101",
               D(6) when "0110",
               D(7) when "0111",
               D(8) when "1000",
               D(9) when "1001",
               D(10) when "1010",
               'X' when others;
```

```
    end function_table_ws;

--  FULL ADDER for Flanger summation

  library IEEE;
  use IEEE.std_logic_1164.all;

  entity FBA_FLANGING is
    port (
      S1  : in std_logic;
      FOUT : in std_logic;
      FCIN : in std_logic;
      FSUM : out std_logic;
      FCOUT: out std_logic;
      AU_LRCK: in std_logic
    );
  end FBA_FLANGING;

  architecture behavioral of FBA_FLANGING is
  FCIN <= 0;                              -- input carry starts at 0;
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      FSUM <= S1 xor FOUT xor FCIN;
      FCOUT <= (S1 and FOUT) or (S1 and FCIN) or (FOUT and FCIN);
    end if;
      FCIN <= FCOUT;
    end process;
  end behavioral;
-- COMPLETE FLANGING UNIT


-- ECHO UNIT attached to O(2) and E(2)

  library ieee;
  use ieee.std_logic_1164.all;
  entity srg_50ms is  -- 50 ms delay
    port(AU_LRCK,  EI : in std_logic;
    QE  : out std_logic_vector(4882 downto 0);
    EO : out std_logic);
  end srg_50ms;

  architecture behavioral of srg_50ms is
  signal shift : std_logic_vector(4882 downto 0);
  begin
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      shift <= shift(4881 downto 0) & EI;
    endif;
  end process;
    QE <= shift;
    EO <= .5*shift(4882);
  end behavioral;

  -- FULL ADDER TO SUM
  library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.STD_LOGIC_ARITH.ALL;

  entity FBA_ECHO is
    port (
      EO : in std_logic;
      EI  : in std_logic;
      ECIN : in std_logic;
      ESUM : out std_logic;
      ECOUT: out std_logic;
      AU_LRCK: in std_logic
    );
  end FBA_ECHO;

  architecture behavioral of FBA_ECHO is
```

```
  ECIN <= 0; -- carry starts at zero
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      ESUM <= EI xor EO xor ECIN;
      ECOUT <= (EO and EI) or (EO and ECIN) or (EI and ECIN);
    end if;
  RCOUT <= RCIN;
  end behavioral;
-- COMPLETE ECHO UNIT


-- REVERB/VIBRATO UNIT attached between O(3) and E(3)

-- 20 ms Delay Unit w/ 5. gain on
  library ieee;
  use ieee.std_logic_1164.all;
  use IEEE.STD_LOGIC_ARITH.ALL;
  entity srg_20ms is  -- 1953 delays = 20 ms delay
    port(AU_LRCK, RSUM : in std_logic;   --
    QR  : out std_logic_vector(1952 downto 0);
    RO : out std_logic);
  end srg_20ms;

  architecture behavioral of srg_20ms is
  signal shift : std_logic_vector(    downto 0);
  begin
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      shift <= shift(1951 downto 0) & RSUM;
    endif;
  end process;
    Q <= shift;
    RO <= .5*shift(1952);
  end behavioral;

  -- FULL ADDER TO SUM
  library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.STD_LOGIC_ARITH.ALL;
  entity FBA_RV is
    port (
      RO : in std_logic;
      RI  : in std_logic;
      RCIN : in std_logic;
      RSUM : out std_logic;
      RCOUT: out std_logic;
      AU_LRCK: in std_logic
    );
  end FBA_RV;

  architecture behavioral of FBA_RV is
  RCIN <=0;                              -- carry starts at zero
  process (AU_LRCK)
  begin
    if (AU_LRCK'event and (AU_LRCK = '1')) then
      RSUM <= RI xor RO xor RCIN;
      RCOUT <= (RO and RI) or (RO and RCIN) or (RI and RCIN);
    end if;
  RCOUT <= RCIN;
  end behavioral;
-- COMPLETE REVERB UNIT

-- 4 to 1 MUX for end of audio effects unit
library ieee;
use ieee.std_logic_1164.all;
entity mux_4_to_1 is
  port (MUX_CONTROL: in std_logic_vector(1 downto 0); -- Control logic for
MUX/DEMUX
        E: in std_logic_vector(0 to 3);
        TONE: out std_logic;)
```

```
  E(0) <= S0;
  E(1) <= FSUM;
  E(2) <= ESUM;
  E(3) <= RO;
end mux_4_to_1;

architecture function_table_ws of mux_4_to_1 is
begin
  with MUX_CONTROL select
     TONE <= E(0) when "00",            --  parallel audio out
             E(1) when "01",
             E(2) when "10",
             E(3) when "11",
             'X' when others;
end function_table_ws;
-- COMPLETE 4 to 1 MUX

-- Tone Control Unit: Hierarchical

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
entity audio_effects is
  port ( AO : out std_logic_vector(2 downto 0);
    TONE, CLK : in std_logic);
end audio_effects;

architecture behavioral of audio_effects is
  oldMid <= 0;
  oldLow <= 0;
begin
process (CLK)
begin
  -- provides 3 audio channels, with fc ~ 238 Hz for mid/low
  if (CLK'event and (CLK = '1')) then
    A(2) <= AI - oldLow - oldMid;
    A(1) <= (A(2)/32) + oldMid;
    A(0) <= (A(1)/32) + oldLow;
  endif;
end process;
  oldMid <= A(1);
  oldLow <= A(0);
  A2 <= A(2);
  A1 <= A(1);
  A0 <= A(0);
end behavioral;

--------------------------------------------------------------------------------
-- NEED TO WRITE THESE VALUES TO A REGISTER or RAM !!! --
--------------------------------------------------------------------------------

--- IMPLEMENT REAL TIME GAIN CONTROL VIA C INTERFACE --
--  NEED OPB, READ IN VALUES, MULTIPLY by volume in C
--  then output back to different location on the register

--------------------------------------------------------------------------------
-- READ 3 INPUTS BACK FROM REGISTER/RAM
--------------------------------------------------------------------------------

-- Sums 3 inputs
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
entity summation_3_inputs is
  port (Y : in std_logic_vector(2 downto 0); -- takes 3 input channels
        POUT : out std_logic); -- 1 output channel
end summation_3_inputs;

architecture function_table of summation_10_inputs is
begin
  POUT <= Y(1) + Y(2) + Y(0);
end function_table;
```

AK4565.vhd heavily based on Christian's:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity audio_ak4565 is

port   (
          CLK : in std_logic;
              RST : in std_logic;

              audio_dacdata : in std_logic;
              audio_adcdata : out std_logic;

              interrupt : out std_logic;
              audio_fifohalf : out std_logic;

              AU_CSN_N : out std_logic;
      AU_BCLK : out std_logic;
      AU_MCLK : out std_logic;
      AU_LRCK : out std_logic;          -- 2x sampling rate clock, 1 16-bit
                                        -- channel/ sample
      AU_SDTI : out std_logic;          -- Digital input to chip
      AU_SDTO0 : in std_logic           -- Digital output from chip
        );
end audio_ak4565;


architecture Behavioral of audio_ak4565 is

signal clkcnt : std_logic_vector(4 downto 0);
signal audiocnt_dac, audiocnt_adc : std_logic_vector(11 downto 0);

signal audio_clk : std_logic;
signal lrck : std_logic;

begin

AU_CSN_N <= '1'; -- no chip select as we don't write the ctrl regs

-- generate the 3 clocks: master, serial, frame

process(CLK, RST)
begin
       if rst = '1' then
              clkcnt <= "00000";
       elsif clk'event and clk='1' then
              clkcnt <= clkcnt + 1;
       end if;
end process;
AU_MCLK <= clkcnt(1); -- master clock, 12.5 MHz
audio_clk <= clkcnt(4); -- this is the serial clock, 1.5625 Mhz
AU_BCLK <= not audio_clk; -- don't ask but read AK4565 specs

process(audio_clk, RST)
begin
       if rst = '1' then
       audiocnt_dac <= "000000000000";
       audiocnt_adc <= "111111111111";
       lrck <= '0';
       elsif audio_clk'event and audio_clk='1' then
              audiocnt_dac <= audiocnt_dac + 1;
              audiocnt_adc <= audiocnt_adc + 1;
              lrck <= not audiocnt_dac(3);
       end if;
```

```
end process;
AU_LRCK <= lrck; -- audio clock, 97.656 KHz.  This was doubled from Christian's
                 -- design by decrementing the bit on which this clock runs,
                 -- making it change twice as fast as originally, thus doubling
                 -- the frequency.

interrupt <= not audiocnt_dac(10);
audio_fifohalf <= audiocnt_dac(11);

audio_bram_addr_dac <= audiocnt_dac;
audio_bram_addr_adc <= audiocnt_adc;
audio_bram_clk <= audio_clk;

AU_SDTI <= audio_dacdata;
audio_adcdata <= AU_SDTO0;
end architecture;


library IEEE;
 use IEEE.STD_LOGIC_ARITH.all;
 use IEEE.STD_LOGIC_UNSIGNED.all;

 entity parallel_to_serial is
   port (
     audio_adcdata : out std_logic;      -- serial input to codec, post FX
     AU_BCLK : in std_logic;            -- serial bit clock
     AU_LRCK : in std_logic;            -- parallel clock
     POUT : in std_logic_vector(15 downto 0)  -- Input from FX/Tone control
   );
  end parallel_to_serial;

 architecture Behavioral of parallel_to_serial is
 count = 0;
 signal shift : std_logic_vector(15 downto 0);
 process (AU_BCLK, AU_LRCK)
 begin
   if AU_BCLK'event and AU_BCLK = '1' then
     audio_adcdata <= shift(count);
     count = count + 1;
   end if;
   if AU_LRCK'event and AU_LRCK = '1'then
     shift(15 downto 0) <= POUT(15 downto 0);
     count = 0;
   end if;
 end process;
 end behavioral;



 library IEEE;
 use IEEE.STD_LOGIC_ARITH.all;
 use IEEE.STD_LOGIC_UNSIGNED.all;

 entity serial_to_parallel is           -- basically a shift-register
   port (
     audio_dacdata : in std_logic;      -- serial input from codec
     AU_BCLK : in std_logic;            -- serial bit clock
     AU_LRCK : in std_logic;            -- parallel clock

     PIN : out std_logic_vector(15 downto 0)        -- output to FX
   );
 end serial_to parallel;

 architecture Behavioral of serial_to_parallel is
 count = 0;
 signal shift : std_logic_vector(15 downto 0);
 process (AU_BCLK)
 begin
   if AU_BCLK'event and AU_BCLK = '1'then
     if count = '1' then
       PIN(15 downto 0) <= shift (15 downto 0);
       count = 0;
```

```
     end if;
     shift <= shift(15 downto 0) & audio_dacdata;
     count = count + 1;
   end if;
  end process;
 end behavioral;
```

C Written

```
#include "xbasic_types.h"
#include "xio.h"
#include "fonts.h" //code for characters in bitmap

//Height Width Dimensions
#define W 640
#define H 480

//Pixel Base Address
#define VGA_START 0x00800000

//Memory Base Address
#define ERF 0xFEFF020C
//#define FADES 06H
#define TONE1A 0xFEFF0200
#define TONE2A 0xFEFF0204
#define TONE3A 0xFEFF0208
#define TONE1B 0xFEFF0210
#define TONE2B 0xFEFF0214
#define TONE3B 0xFEFF0218
//Character dimensions
#define FONT_HEIGHT 8
#define FONT_WIDTH 8


#define RED 0xE0
#define BLUE 0x03



void write_char (char x, int oset) //writes char to screen
{
  int i=0;
  int M=1;

       for (i=0; i<8; i++)
         {
           if((M&x)==0)
           {
             XIo_Out8(VGA_START+ (8-i)+oset, RED|BLUE);
           }
           else
           {
             XIo_Out8(VGA_START+ (8-i)+oset, BLUE);
           }
           M=M<<1;
         }

}

void print_char (char z, int x, int y)
{
  int i=0;
  for (i=0; i<8; i++)
    {
      write_char(fontcode[z*8+i], x*8+(y+i)*640);
    }
}
```

```
void print_string(char* z, int len_str, int x, int y)//prints string
{
  int i=0;
  for (i=0; i<len_str; i++)
    {
      print_char (z[i],x+i, y);
    }
}

void clear()
{
  int x;

for(x=0; x<307200; x++) //sets the screen to purple background
    {
      XIo_Out8(VGA_START +x, RED|BLUE);
    }
}

int  menuselect(int exit)
 {
   int choice;
   // print_string("Please enter number of choice",29,100,100);

   print_string("Please enter number: 1. Echo", 28, 100,110);

   print_string("2. Reverb", 9,121,120);

   print_string("3. Flanging", 11,121,130);

   print_string("4. Default",10 ,121,140);

   /*   scanf("%d", &choice);
   if (choice==8) return 1;
   switch (choice)
     {
     case 1: echo(); break;
     case 2: reverb();break;
     case 3: flang();break;
     case 4: fadein();break;
     case 5: fadeout();break;
     case 6: tone();break;
     case 7: {ERF<<0;FADES<<0;} break;
     }*/
     return 0;
 }

/*
echo()
{
  ERF<<64;
}

reverb()
{
  ERF<<192;
}

flang()
{
  ERF<<128;
}

fadein()
{
  FADES<<4;
}

fadeout()
{
  FADES<<2;
```

```
}

int toneOut(int xval,int vn)
{
  return xval*pow(2,((vn/2)-4));
}


tone()
{

  int xin, xmid, xlow;
  int value, choice;
  xin=TONE1A;
  xmid=TONE2A;
  xlow=TONE3A;
  clear();
  print_string("choose:: 1. xn 2.xmid 3.xlow",28, 200,200 )
    scanf("%d", &choice);
  clear();
  print_string("enter value from 0 to 15"), 24, 200,200);
    scanf("%d", &value);
    switch (choice)
       {
       case 1:
         TONE1B<< toneOut(xin, value);
       case 2:
         TONE2B<<toneOut(xmid, value);
       case 3:
         TONE3B<<toneOut(xlow, value);
       }

}

*/
main()
{

int exit, temp;
exit=0;
clear ();
while (exit!=1)
{
temp=menuselect(exit);
exit=temp;
}
}
```