
Team: Adegoke Adediran
Neil Sarkar
Stephanie Maryon

D*Voice*R

D i g i t a l   V o i c e   R e c o r d e r

Table of Contents

1 Introduction

Our project is a digital voice recorder. The voice recorder is accessed with a user interface through a CPU. The user interface digitally records and playbacks sound, specifically a voice, on one of two chosen tracks for about sixteen seconds via a microphone, speakers, and the board. The microphone is connected to the int0 pin on the board and used to record audio data to the XESS board. A user can then play back one of their recordings and listen to it through a speaker connected to the RCA jack. The components used on the board are the Xilinx Spartan-IIE 1.8V FPGA, the AKM AK4565, the Toshiba TC55V16256J 256k X 16 SRAM, and the OPB bus.

2 Design Components

2.1 Audio Codec: AKM AK4565:

The AKM AK4565 is an audio codec which enables us to connect a microphone and speakers to our system. The audio codec had built in analog to digital and digital to analog converter, so we use it to accept an analog signal from the microphone in order to store the bits in a FIFO, or to receive a digital stream of bits from the CPU, which are stored in a FIFO and send the converted analog signal out through speakers. The Intr0 pin is used to connect the microphone to the codec and the Out outputs to connect the speakers.

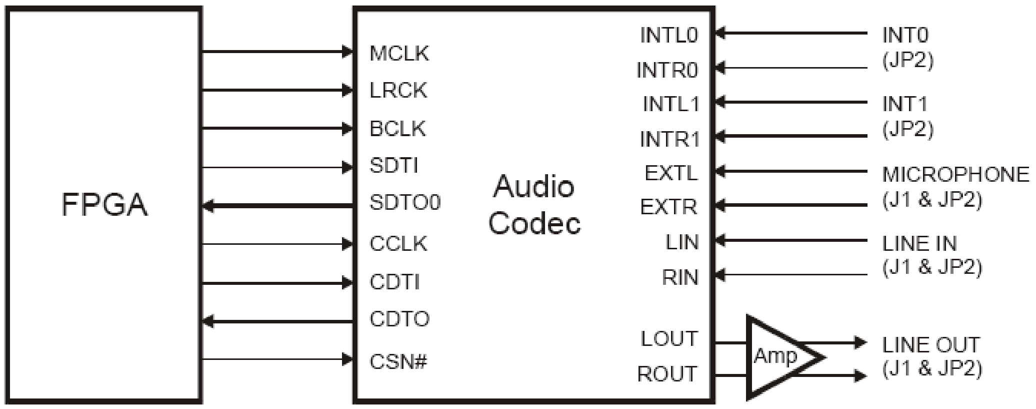

Fig 1. FPGA connection to the Audio Codec

The audio codec uses three clocks to coordinate its system, a master clock, a clock for audio serial data, and a channel clock. We made two FIFOs, two shift registers, input and output registers, and a finite state machine for the codec. We used counters to implement the clocks. Data is put into or taken out of the FIFOs by the codec every channel clock event that is being read or written to by the OPB bus. The shift registers shift every serial data clock event.

One of the challenges of implementing our codec was synchronization between the OPB bus and the codec. Our first idea was to use BRAMs to interface the data between the OPB bus and the codec. We were able to send data back and forth, but the data was random. This was because the addresses of the BRAM were not synchronized. We could not figure out a proper coordination scheme. Then we were thinking about using a polling system to locate the instance the data was ready to be either read or written. This seemed too simple and slow for the system that we wanted to design, so we decided to use FIFOs. This eliminated the address problem of the BRAMs while keeping the speed of our system up by using a memory interface between the OPB bus and the codec.

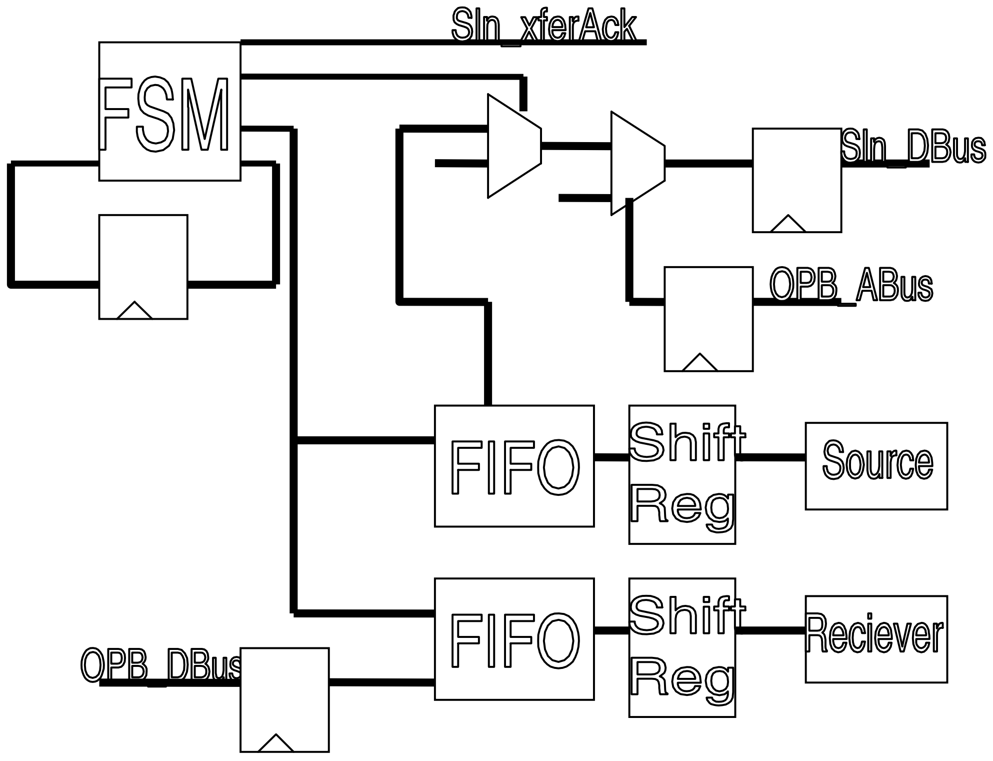

Fig 2. Audio Codec Datapath

The datapath of the codec is shown above. Either a read or a write an analog input signal comes from the source and is converted to a digital signal. The digital signal goes into a shift register to form a 16-bit data value. The data value is put into a FIFO and waits for retrieval from the OPB bus. The codec receives control from the finite state machine telling it to either write or read a value from the FIFO. The rightmost multiplexor is used to poll the codec to find out if the FIFOs have data if the OPB bus wants to read data and record a value, or if the FIFOs are not full so that data can be written and played back through the audio codec. The finite state machine is shown below in figure 3. The cuss signal is a chip select determined by the top 16-bits of the

OPB_ABus and the OPB_Select signal. The address of the codec is 0xFEFE0000 for writing and reading, and is 0xFEFE0004 for polling.

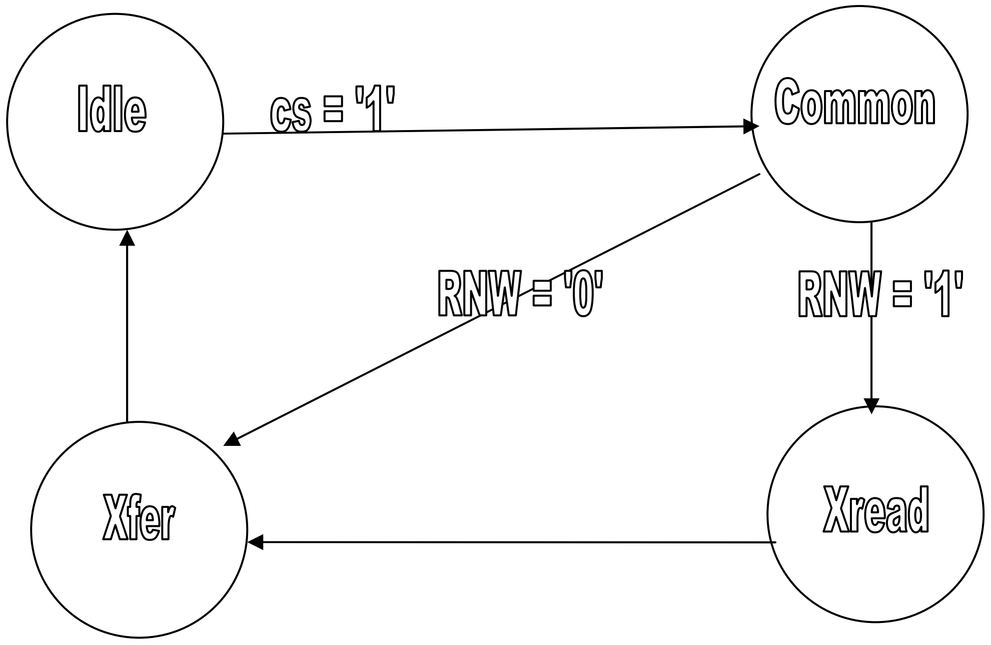

Fig 3. Audio Codec Finite State Machine

2.2 The Toshiba TC55V16256J 256k X 16 SRAM:

The SRAM is used to hold the samples we feed into the codec from the microphone. It is organized as a 262,144 by 16-bit array of memory blocks. Each sample received from the codec 16 bits long. Therefore our memory will be able to hold 262,144 samples. We chose to design our system based on an 8 KHz sample rate. Every second 96,000 samples are stored in memory by the codec. (We are sampling from both the left and the right channels). If we write one out of every eight samples to memory, we will achieve our sample rate of 8 KHz. This will allow us to store a total of approximately 16 seconds. Our SRAM is divided into two tracks in our user implementation file. Each track allows the user to store an 8 second long audio clip. The basic signals used to instantiate the SRAM are shown in figure 4.

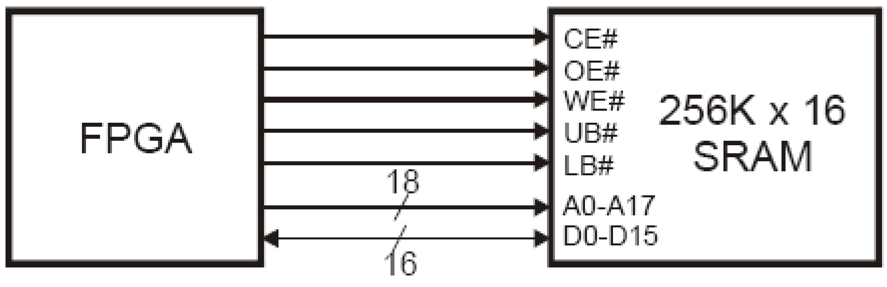

Fig 4: FPGA connection to the SRAM

The datapath of our SRAM is shown below. It includes multiple input and output registers, a finite state machine, and an input output buffering system as the data wires to the SRAM are both inputs and outputs. A request is sent by the OPB bus to either read or write to the SRAM. The finite state machine sees the request and sends the proper controls to the SRAM which tell it to read or write data. The cs signal, which is the chip select is implemented in the same way respectively as it in the codec. The chip is selected when OBP_Select is high and when the address of the SRAM is selected, which means that the top twelve bits of the OPB_ABus will be high. The SRAM addresses start at 0xFFF00000 and continue to 0xFFF3FFFF. The finite state machine moves out of idle, it moves into a common mode which is not dependent on reading or writing. If the access is a read the SRAM reads the codec and transmits it in the xmit state. If the access is a write, the SRAM can just move into the xmit state and transfer the data to be written.

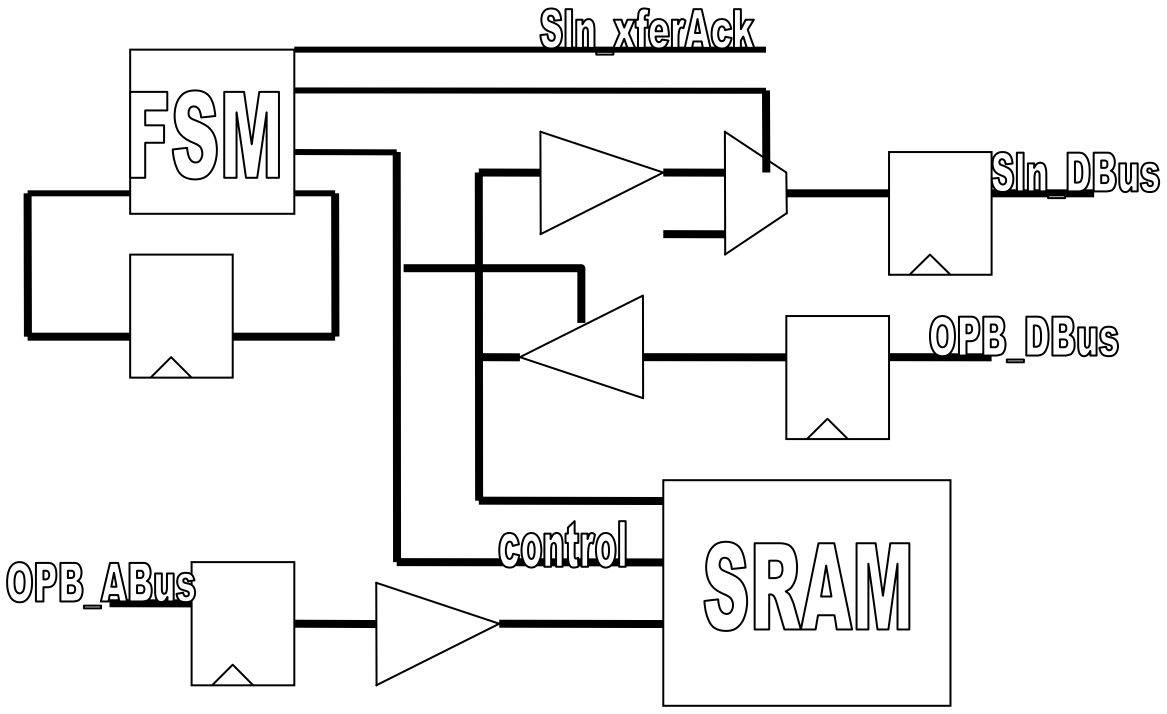

Fig 5. SRAM Datapath

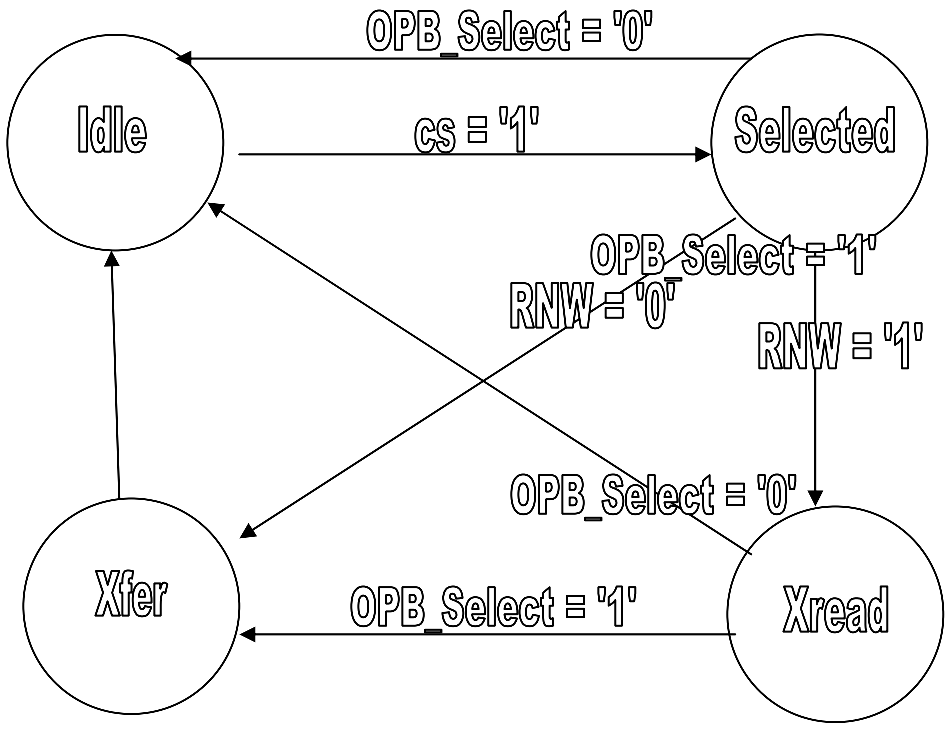

Fig 6. SRAM Finite State Machine

Once we were successfully reading and writing data, we started thinking about file system implementation. With the size of the files, our original idea of a primitive, variable length file system seemed unnecessary, as a user would realistically only want about 5 files, as there is only 32 seconds of audio to divide amongst the files. Originally, we tried to implement a four file system, but there were some memory issues with the SRAM, so we defaulted to having two hard-coded files in order to have a functioning demo.

2.3 The User Interface

Our user interface is somewhat minimal but an essential part to our project. It is used not only as a means of control for the user, but it also polls the codec to see if data can be read or written and also controls the sampling rate of our system. Requests are made via the keyboard, and transmitted to the CPU through interrupts that are handled using the UART. '1', '2', 'r', and 'p' can be entered by the user to choose tracks 1 or 2 and to record and play data. One issue we had with our main c program was that we ran out of room to store the file in the BRAM. Originally we had four tracks in the SRAM. For reasons of lack of time we decided instead of using space in the SRAM for our code overflow, to decrease the size of the code by decreasing the number of tracks down to two.

3. Conclusions

3.1 Further Developments

There were a few features we would have liked to add, but were not able to because we did not have enough time. If we had a few more weeks, here's a list of features and functionalities that we would have liked to implement, in order of importance:

-A data compression algorithm
-Flash memory replacing the SRAM for storage
-Variable Length File Sizes
-Variable Sampling Rate
-Voice Modification/Equalizing Effects

3.2 Lessons Learned

Stephanie Maryon:

For this project I wrote the vhdl hardware code for the SRAM and for the audio codec with the help of the TA Christian, who is very helpful and intelligent. I also wrote the user interface of the project. I learned a lot doing this project because I did a lot of it and spent many, many hours on it. I learned how an OBP bus is suppose to interact with different components without many stalls so the speed of the system is not derogated. I learned about an audio codec and how to implement it. There are a lot of different ways a codec can be used and configured. I learned how hard VHDL is to implement, especially because of debugging problems which is why I like VLSI and layout so much better. I like the idea of custom circuits that you build blueprints for and know exactly what they do, rather then having to write vhdl code in which it is very hard to know exactly what is going on. I am more of a visual person. I also was able to brush up on my c code with this project as I am a hardware engineer and hate writing software code.

Adegoke Adediran:

I have learned a great deal from this project. Being fairly new to hardware design, this project gave me an opportunity to understand and assimilate the capabilities of VHDL. The challenge the project posed was rather exciting, although the task of deciphering VHDL code can be daunting. I have a new appreciation for system designers, as synchronizing clock signals proved to be a very difficult task for me. I also got some new insights about the C programming language.

Neil Sarkar:

VHDL
The VHDL is hands down the most challenging part of the course, and it will certainly be the difference between a working and a non-working project.

One thing to make sure of is that the group is comfortable with and can explain the VHDL solutions to the later labs. When a lab works, its very tempting to say “Well, that fixed it...but I'm not sure why. Oh well.” However, this will only come back to bite you when you have to design a peripheral from scratch.

Make sure that the majority of your group is proficient in VHDL. Otherwise, the vast majority of the work will fall to the minority of the group, which is not only stressful for those proficient in VHDL, but also frustrating to those that are not.

Old Projects
Perusing through old projects early on (weeks leading up to project proposal) is a very good idea that we did not capitalize on enough. It gives you a much better idea of what the board is and is not capable of doing, and also gives you more of a chance to pick a project subject and scope that will be both feasible and engaging.

Similarly, looking at code of groups that did a project that resembles yours during the very early phases of your project development is second to none in terms of initial guidance and getting a foothold on the challenges that you will face.

However, it's definitely a bad idea to port code wholesale from other groups, especially VHDL. What another group hacked in before their project was due is not likely to be extensible to deal with your project. More importantly, it's crucial to fully understand every line of VHDL that is included in your project.

4. Appendix (Code Index)

system.ucf
```
net sys_clk period = 18.000;

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
net RS232_RD loc="p73";

net PB_D<0> loc="p153";
net PB_D<1> loc="p145";
net PB_D<2> loc="p141";
net PB_D<3> loc="p135";
net PB_D<4> loc="p126";
net PB_D<5> loc="p120";
net PB_D<6> loc="p116";
net PB_D<7> loc="p108";
net PB_D<8> loc="p127";
net PB_D<9> loc="p129";
net PB_D<10> loc="p132";
net PB_D<11> loc="p133";
net PB_D<12> loc="p134";
net PB_D<13> loc="p136";
net PB_D<14> loc="p138";
net PB_D<15> loc="p139";

net PB_A<0> loc="p83";
net PB_A<1> loc="p84";
net PB_A<2> loc="p86";
net PB_A<3> loc="p87";
net PB_A<4> loc="p88";
net PB_A<5> loc="p89";
net PB_A<6> loc="p93";
net PB_A<7> loc="p94";
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";

net PB_CE loc="p147";
net PB_OE loc="p125";
net PB_WE loc="p123";
net PB_UB loc="p146";
net PB_LB loc="p140";

net AU_CSN loc="p165";
net AU_MCLK loc="p167";
net AU_LRCK loc="p168";
net AU_BCLK loc="p166";
net AU_SDTI loc="p169";
net AU_SDTO loc="p173";
```

system.mhs
```
# Parameters
PARAMETER VERSION = 2.1.0

# Global Ports

PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN

PORT PB_D = PB_D, DIR = INOUT, VEC=[15:0]
PORT PB_A = PB_A, DIR = OUT, VEC=[17:0]
PORT PB_WE = PB_WE, DIR = OUT
PORT PB_OE = PB_OE, DIR = OUT
PORT PB_LB = PB_LB, DIR = OUT
PORT PB_UB = PB_UB, DIR = OUT
PORT PB_CE = PB_CE, DIR = OUT
PORT AU_CSN = AU_CSN,              DIR=OUT
PORT AU_BCLK         = AU_BCLK,           DIR=OUT
PORT AU_MCLK         = AU_MCLK ,          DIR=OUT
PORT AU_LRCK         = AU_LRCK,           DIR=OUT
PORT AU_SDTI         = AU_SDTI,           DIR=OUT
PORT AU_SDTO         = AU_SDTO,           DIR=OUT

BEGIN opb_bram
  PARAMETER INSTANCE = bram_peripheral
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_BASEADDR = 0xFFF00000
  PARAMETER C_HIGHADDR = 0xFFF3FFFF
  PORT OPB_Clk = sys_clk
  BUS_INTERFACE SOPB = myopb_bus
  PORT PB_D = PB_D
  PORT PB_A = PB_A
  PORT PB_WE = PB_WE
  PORT PB_OE = PB_OE
  PORT PB_LB = PB_LB
  PORT PB_UB = PB_UB
  PORT PB_CE = PB_CE
END

BEGIN opb_ak4565
  PARAMETER INSTANCE = ak4565_peripheral
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_BASEADDR = 0xFEFE0000
  PARAMETER C_HIGHADDR = 0xFEFEFFFF
  PORT OPB_Clk = sys_clk
  BUS_INTERFACE SOPB = myopb_bus
  PORT AU_CSN        = AU_CSN
  PORT AU_BCLK       = AU_BCLK
  PORT AU_MCLK       = AU_MCLK
  PORT AU_LRCK       = AU_LRCK
  PORT AU_SDTI       = AU_SDTI
  PORT AU_SDTO       = AU_SDTO
END

# Interrupt controller for dealing with interrupts from the UART
```

```
BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFEFF0000
 PARAMETER C_HIGHADDR = 0xFEFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr = uart_intr
 PORT IRq = intr
 BUS_INTERFACE SOPB = myopb_bus
END

# The main processor core
BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.
# Data LMB bus
BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus
BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory
BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run
BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

# The OPB bus controller connected to the Microblaze
BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

# UART: Serial port hardware
BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT Interrupt = uart_intr
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END
```

system.mss

```
 PARAMETER VERSION = 2.2.0
 PARAMETER HW_SPEC_FILE = system.mhs

BEGIN OS
 PARAMETER PROC_INSTANCE = mymicroblaze
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 1.00.a
 PARAMETER STDIN = myuart
 PARAMETER STDOUT = myuart
END

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = intc
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.c
END

# Use null drivers for peripherals that don't need them
# This supresses warnings
BEGIN DRIVER
 PARAMETER HW_INSTANCE = bram_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = ak4565_peripheral
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_data_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_instruction_controller
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END
```

opb_bram_v2_1_0.mpd

```
###################################################################
##
## Microprocessor Peripheral Definition
##
###################################################################

BEGIN opb_bram

OPTION IPTYPE = PERIPHERAL
OPTION EDIF=TRUE

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],     BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN,                                 BUS = SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],     BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,                                 BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,                                 BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,                                 BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,                                 BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,                                BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,                                BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,                                BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,                                BUS = SOPB
PORT PB_D      = "",          DIR=INOUT,     VEC=[15:0],          3STATE=FALSE,
IOB_STATE=BUF
PORT PB_A      = "",          DIR=OUT,       VEC=[17:0],          IOB_STATE=BUF

PORT PB_WE     = "",          DIR=OUT
PORT PB_OE     = "",          DIR=OUT
PORT PB_LB     = "",          DIR=OUT
PORT PB_UB     = "",          DIR=OUT
PORT PB_CE     = "",          DIR=OUT


END
```

opb_bram.vhd

```
-------------------------------------------------------------------------------
--
-- Simple OPB peripheral: a BRAM controller
--
-- Embedded Systems
-- Columbia University
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity opb_bram is

  generic (
    C_OPB_AWIDTH : integer                  := 32;
    C_OPB_DWIDTH : integer                  := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;
    PB_D        : inout std_logic_vector(15 downto 0);
    PB_A        : out std_logic_vector(17 downto 0);
    PB_WE       : out std_logic;
    PB_OE       : out std_logic;
    PB_LB       : out std_logic;
    PB_UB       : out std_logic;
    PB_CE       : out std_logic);       -- Transfer acknowledge



end opb_bram;

architecture Behavioral of opb_bram is

  constant RAM_AWIDTH : integer := 18;  -- Number of address lines on the RAM
  constant RAM_DWIDTH : integer := 16;  -- Number of data lines on the RAM

 component OBUF_F_24
    port (
    O : out std_logic;
    I : in std_logic);
  end component;

  component IOBUF_F_24
   port (
    O : out std_logic;
    IO : inout std_logic;
    I : in std_logic;
    T : in std_logic);
  end component;

  signal RNW : std_logic;
  signal chip_select : std_logic;
  signal output_enable : std_logic;
  signal tri_iob: std_logic;
  signal OE, WE, LB, UB : std_logic;
  signal saddin : std_logic_vector(0 to 17);
  signal sramout : std_logic_vector(0 to 15);
  signal sramin : std_logic_vector(0 to 15);
  signal iobuf : std_logic;


type opb_state is (Idle, Selected, Read, Xfer);
signal present_state, next_state : opb_state;

begin


  --generate buffers for address and data
  address: for i in 0 to 17 generate
  OBUFBlock  : OBUF_F_24
   port map (
    O => PB_A(i),
    I  => saddin(i));
  end generate;

  data : for j in 0 to 15 generate
  IOBUFBlock : IOBUF_F_24
   port map (
    O => sramout(j),
    IO => PB_D(j),
    I => sramin(j),
    T => tri_iob);
  end generate;


  --registers for opb inputs
  register_opb_inputs: process (OPB_Clk, OPB_Rst)
  begin
   if OPB_Rst = '1' then
    sramin <= (others => '0');
    saddin <= (others => '0');
    RNW <= '0';
   elsif OPB_Clk'event and OPB_Clk = '1' then
    sramin <= OPB_DBus(0 to RAM_DWIDTH-1);
    saddin <= OPB_ABus(C_OPB_AWIDTH-3-(RAM_AWIDTH-1) to C_OPB_AWIDTH-3);
    RNW <= OPB_RNW;
  end if;
end process register_opb_inputs;

--registers for opb outputs
register_opb_outputs: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');

  elsif OPB_Clk'event and OPB_Clk = '1' then
   if output_enable = '1' then
      Sln_DBus(0 to RAM_DWIDTH-1) <= sramout;
   else
     Sln_DBus(0 to RAM_DWIDTH-1) <= (others => '0');
   end if;
  end if;
end process register_opb_outputs;

--combinational signals
Sln_errAck  <= '0';
Sln_retry   <= '0';
Sln_toutSup <= '0';
Sln_DBus(RAM_DWIDTH to C_OPB_DWIDTH-1) <= (others => '0');

chip_select <=
  '1' when OPB_select = '1' and
      OPB_ABus(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) =
      C_BASEADDR(0 to C_OPB_AWIDTH-3-RAM_AWIDTH) else
  '0';

PB_CE <= '0';
PB_WE <= WE;
PB_OE <= OE;
PB_UB <= UB;
PB_LB <= LB;

-- Sequential part of the FSM
fsm_seq : process(OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    present_state <= Idle;
  elsif OPB_Clk'event and OPB_Clk = '1' then
    present_state <= next_state;

  end if;
end process fsm_seq;

-- Combinational part of the FSM
fsm_comb : process(OPB_Rst, present_state, chip_select, OPB_Select, RNW)
begin
  -- Default values
  Sln_xferAck <= '0';
  next_state <= present_state;
  output_enable <= '0';
```

```
    tri_iob <= '1';
    WE <= '1';
    UB <= '0';
    LB <= '0';
    OE <= '1';

    case present_state is

      when Idle =>
        if chip_select = '1' then
          next_state <= Selected;
        end if;

      when Selected =>
        if OPB_Select = '1' then
          if RNW = '1' then
            OE <= '0';
            next_state <= Read;
          else
            WE <= '0';
            tri_iob <= '0';
            next_state <= Xfer;
          end if;
        else
          next_state <= Idle;
        end if;

      when Read =>

        if OPB_Select = '1' then
          OE <= '0';
          output_enable <= '1';
          next_state <= Xfer;
        else
          next_state <= Idle;
        end if;

      when Xfer =>
        Sln_xferAck <= '1';
        next_state <= Idle;

     end case;

  end process fsm_comb;

  end Behavioral;
```

opb_ak4565_v2_1_0.mpd

```
###################################################################
##
## Microprocessor Peripheral Definition
##
###################################################################

BEGIN opb_ak4565

OPTION IPTYPE = PERIPHERAL
OPTION EDIF=TRUE

OPTION STYLE = MIX

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

## Generics for VHDL
PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE = 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
PARAMETER c_opb_dwidth   = 32,         DT = integer

## Ports
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [0:(c_opb_awidth-1)],     BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [0:((c_opb_dwidth/8)-1)], BUS = SOPB
PORT opb_clk     = "",          DIR = IN,                                 BUS = SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [0:(c_opb_dwidth-1)],     BUS = SOPB
PORT opb_rnw     = OPB_RNW,     DIR = IN,                                 BUS = SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,                                 BUS = SOPB
PORT opb_select  = OPB_select,  DIR = IN,                                 BUS = SOPB
PORT opb_seqaddr = OPB_seqAddr, DIR = IN,                                 BUS = SOPB
PORT sln_dbus    = Sl_DBus,     DIR = OUT, VEC = [0:(c_opb_dwidth-1)],    BUS = SOPB
PORT sln_errack  = Sl_errAck,   DIR = OUT,                                BUS = SOPB
PORT sln_retry   = Sl_retry,    DIR = OUT,                                BUS = SOPB
PORT sln_toutsup = Sl_toutSup,  DIR = OUT,                                BUS = SOPB
PORT sln_xferack = Sl_xferAck,  DIR = OUT,                                BUS = SOPB

PORT AU_CSN = "",              DIR=OUT
PORT AU_BCLK         = "",          DIR=OUT
PORT AU_MCLK         = "",          DIR=OUT
PORT AU_LRCK         = "",          DIR=OUT
PORT AU_SDTI         = "",          DIR=OUT
PORT AU_SDTO         = "",          DIR=IN

END
```

opb_ak4565.vhd

```
-------------------------------------------------------------------------------
--
-- Simple Ak4565 peripheral: a Codec controller
--
-- Embedded Systems
-- Columbia University
--
-------------------------------------------------------------------------------
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 opb_ak4565 is

  generic (
    C_OPB_AWIDTH : integer                   := 32;
    C_OPB_DWIDTH : integer                   := 32;
    C_BASEADDR   : std_logic_vector(0 to 31) := X"00000000";
    C_HIGHADDR   : std_logic_vector(0 to 31) := X"FFFFFFFF");

  port (
    OPB_Clk     : in  std_logic;
    OPB_Rst     : in  std_logic;
    OPB_ABus    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW     : in  std_logic;
    OPB_select  : in  std_logic;
    OPB_seqAddr : in  std_logic;        -- Sequential Address
    Sln_DBus    : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sln_errAck  : out std_logic;        -- (unused)
    Sln_retry   : out std_logic;        -- (unused)
    Sln_toutSup : out std_logic;        -- Timeout suppress
    Sln_xferAck : out std_logic;

    AU_CSN      : out std_logic;
    AU_BCLK     : out std_logic;
    AU_MCLK     : out std_logic;
    AU_LRCK     : out std_logic;
    AU_SDTI     : out std_logic;
    AU_SDTO     : in  std_logic);

end opb_ak4565;

architecture behavioral of opb_ak4565 is
  component fifo_256x16
    port (
      din: IN std_logic_VECTOR(15 downto 0);
      wr_en: IN std_logic;
      wr_clk: IN std_logic;
      rd_en: IN std_logic;
      rd_clk: IN std_logic;
      ainit: IN std_logic;
      dout: OUT std_logic_VECTOR(15 downto 0);
      full: OUT std_logic;
      empty: OUT std_logic);
  end component;

signal count3 : std_logic_vector(11 downto 0);
signal count : std_logic_vector(4 downto 0);
signal shiftin, shiftout, dshiftin : std_logic_vector(15 downto 0)
:=X"0000";
signal rec, play, audio_bram_clk,audio_bram_adcdata : std_logic;
signal adcdone, dacload, wr : std_logic;
signal dacout, adcout, adc_mux : std_logic_vector(15 downto 0);
signal addr, wdata, rdata : std_logic_vector(31 downto 0);
signal wr_bram : std_logic;
signal addr_bramadc, addr_bramdac : std_logic_vector(7 downto 0);
signal dac_full, adc_empty : std_logic;
signal adc_full, dac_empty : std_logic;
signal cs, rnw, rd_bram : std_logic;

signal bclk,mclk,lrclk : std_logic;


type opb_state is (IDLE, COMMON, XREAD, XFER);
signal cstate, nxstate : opb_state;

signal ld_sldbus : std_logic;

begin  -- behavioral

-- opb stuff

process(OPB_Rst, OPB_Clk)
begin
  if OPB_CLk'event and OPB_clk='1' then
    cstate <= nxstate;
    wdata <= OPB_DBus;
    addr <= OPB_ABus;
    if ld_sldbus = '1' then
      Sln_DBus(0 to 15) <= X"0000";
      Sln_DBus(16 to 31) <= adc_mux;
    else
      Sln_DBus <= (others => '0');
    end if;
    rnw <= OPB_RNW;
  end if;
  if OPB_Rst = '1' then
    cstate <= IDLE;
    wdata <= (others => '0');
    Sln_DBus <= (others => '0');
  end if;
end process;
 adc_mux <= adcout when addr(2 downto 0) = 0
          else X"000" & dac_full & dac_empty & adc_full & adc_empty;
cs <= '1' when OPB_ABus(0 to 15) = C_BASEADDR(0 to 15) and OPB_Select =
'1' else '0';
```

```
-- DAC FIFO
dac_fifo : fifo_256x16
  port map (
    din => wdata(15 downto 0),
    wr_en => wr_bram,
    wr_clk => OPB_Clk,
    rd_en => dacload,
    rd_clk => count(4),
    ainit => OPB_Rst,
    dout => dacout,
    full => dac_full,
    empty => dac_empty
    );

--ADC FIFO
adc_fifo : fifo_256x16
  port map (
    din => rdata(15 downto 0),
    wr_en => wr,
    wr_clk => bclk,
    rd_en => rd_bram,
    rd_clk => OPB_Clk,
    ainit => OPB_Rst,
    dout => adcout,
    full => adc_full,
    empty => adc_empty
    );

process(cstate, cs, rnw)
begin
  nxstate <= cstate;
  wr_bram <= '0';
  Sln_xferAck <= '0';
  ld_sldbus <= '0';
  rd_bram <= '0';

  case cstate is
    when IDLE =>
      if cs = '1' then
        nxstate <= COMMON;
      end if;
    when COMMON =>
      if rnw = '0'  then
        wr_bram <= '1';
        nxstate <= XFER;
      else
        nxstate <= XREAD;
        if addr(2 downto 0) = 0 then
          rd_bram <= '1';
        end if;
      end if;

    when XREAD => ld_sldbus <= '1';
                  nxstate <= XFER;

    when xfer => Sln_xferAck <= '1';
                 NXSTATE <= IDLE;
  end case;


end process;




AU_MCLK <= mclk;
AU_BCLK <= bclk;
AU_LRCK <= lrclk;
AU_CSN <= '1';

rec <= OPB_select and OPB_RNW;
--wr_bram <= OPB_select and not OPB_RNW;
addr_bramadc <= count3(11 downto 4) - 1;
addr_bramdac <= count3(11 downto 4) + 1;

    Sln_errAck <= '0';
    Sln_retry <= '0';
    Sln_toutSup <= '0';
--    Sln_xferAck <= '0';


-- CLOCK Generators  --------------------------------------------------
-----

process(OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      count <= "00000";
    elsif OPB_Clk'event and OPB_Clk = '1'  then
      count <= count + 1;
    end if;
end process;
mclk <= count(1);
bclk <= count(4);

  process(OPB_Rst, bclk)
  begin
    if OPB_Rst = '1' then
      count3 <= X"000";
      lrclk <= '0';
    elsif bclk'event and bclk = '0' then
      count3 <= count3 + 1;

      if count3(4 downto 0) = 31 then
        lrclk <= '1';
      elsif count3(4 downto 0) = 15 then
        lrclk <= '0';
      end if;

      if count3(3 downto 0) = 15  then
        wr  <= '1';
      else
        wr <= '0';
      end if;

    end if;
  end process;





process(OPB_Rst, count(4))
  begin
    if OPB_Rst = '1'  then
      shiftin <= X"0000";
      rdata <= X"00000000";
    elsif count(4)'event and count(4) = '0' then
      if count3(3 downto 0) = 0 then
        rdata(15 downto 0) <= shiftin;
        rdata(31 downto 16) <= X"0000";
      end if;
        shiftin(15) <= shiftin(14);
        shiftin(14) <= shiftin(13);
        shiftin(13) <= shiftin(12);
        shiftin(12) <= shiftin(11);
        shiftin(11) <= shiftin(10);
        shiftin(10) <= shiftin(9);
        shiftin(9) <= shiftin(8);
        shiftin(8) <= shiftin(7);
        shiftin(7) <= shiftin(6);
        shiftin(6) <= shiftin(5);
        shiftin(5) <= shiftin(4);
        shiftin(4) <= shiftin(3);
        shiftin(3) <= shiftin(2);
        shiftin(2) <= shiftin(1);
        shiftin(1) <= shiftin(0);
        shiftin(0) <= AU_SDTO;
    end if;
  end process;

  process(OPB_Rst, count(4))
  begin
    if OPB_Rst = '1' then
      shiftout <= X"0000";
      AU_SDTI <= '0';
    elsif count(4)'event and count(4) = '0' then
      dacload <= '0';

      if count3(3 downto 0) = 14 then
        dacload <= '1';
      end if;

      if count3(3 downto 0) = 14 then
        shiftout <= dacout;
      else
        shiftout <=  shiftout(14 downto 0)  & '0';
      end if;

      AU_SDTI <= shiftout(15);
    end if;

  end process;


end behavioral;
```

main.c

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
```

```
#include "math.h"
#include "xintc_l.h"
#include "xuartlite_l.h"
#define  BTM1 0xFFF20000
#define  CHECK 0xFEFE0004
#define  CODEC 0xFEFE0000
#define  BTM2 0xFFF00000
#define  LEN1 131072/4
#define  LEN2 131072/4


unsigned volatile char buff;

/*
* Interrupt service routine for the UART
*/

void uart_handler(void *callback)
{
  Xuint32 IsrStatus;
  Xuint8 incoming_character;
  /* Check the ISR status register so we can identify the interrupt source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);
  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL | XUL_SR_RX_FIFO_VALID_DATA)) != 0)
  {
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );
      buff= incoming_character;
  }
}

char rd_char()
{

  unsigned char t;
  do{
     t = buff;
   }
  while( t == 0xff);
  buff = 0xff;
  return t;
}


void record(Xuint32 addr, int len)
{
  int i,j;
  Xuint32 x;

  for(i=0;i<len;i++){
    for(j=0;j<16;j++)
      {
      while(XIo_In32(CHECK) & 0x1);
      x=XIo_In32(CODEC);
    }
    XIo_Out16(addr + (i<<2), x);
  }
}

void playback(Xuint32 addr, int len)
{
  int i,j;
  Xuint32 x;

  for(i=0;i<len;i++){
     x = XIo_In16(addr +(i<<2));
     for(j=0;j<16;j++){
            while(XIo_In32(CHECK) & 0x8);
            XIo_Out16(CODEC, x);
     }
  }
}


int main() {

  int i,j ;
  char  key;
  Xuint32 x,y;
  /* Enable UART interrupts and register uart_handler as the ISR */

  XIntc_RegisterHandler( XPAR_INTC_BASEADDR, XPAR_MYUART_DEVICE_ID,
                    (XInterruptHandler)uart_handler, (void *)0);
  XIntc_mEnableIntr( XPAR_INTC_BASEADDR, XPAR_MYUART_INTERRUPT_MASK);
  XIntc_mMasterEnable( XPAR_INTC_BASEADDR );
  XIntc_Out32(XPAR_INTC_BASEADDR +
XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK);
  microblaze_enable_interrupts();
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);


  print("Clearing SRAM\r\n");
  for (i=0; i< LEN1 ; i++)
  {
    XIo_Out16( BTM1+(i<<2) , 0 );
  }
  for (i=0; i<  LEN2 ; i++)
  {
    XIo_Out16(BTM2+(i<<2), 0 );
  }

  print("Clearing done\r\n");

   print("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
   print("\r\n\tWELCOME TO THE DIGITAL VOICE RECORDER\r\n\n\n");
   print("Select which track you want to record/play:\r\n\n");
   print(" 1 - Track 1\r\n");
   print(" 2 - Track 2\r\n");
   print(" Once you select a track, select record or play\r\n");
   print(" p - PLAY\r\n");
   print(" r - REC\r\n");
  //loop to play, record, and select tracks 1 and 2
 while(1){
  key = rd_char();
  while(1){
    while(key != '2' || key == '1'){
      if(key == '1')
             print(" You selected track 1\r\n");
      if(key == 'p')
             print(" PLAY of track 1 ended\r\n");
      if(key == 'r')
             print(" REC of track 1 ended\r\n");
      key = rd_char();
      if( key == 'p')
             {
               print("You selected to PLAY track 1\r\n");
               playback(BTM1, LEN1);
             }
      if( key == 'r')
             {
               print("You selected to RECORD track 1\r\n");
               record(BTM1, LEN1);
             }
    }
    while(key != '1' || key == '2'){
      if(key == '2')
             print(" You selected track 2\r\n");
      if(key == 'p')
             print(" PLAY of track 2 ended\r\n");
      if(key == 'r')
             print(" REC of track 2 ended\r\n");
      key = rd_char();
      if( key == 'p')
             {
               print("You selected to PLAY track 2\r\n");
               playback(BTM2, LEN2);
             }
      if( key == 'r')
             {
               print("You selected to RECORD track 2\r\n");
               record(BTM2, LEN2);
             }
    }
  }
 }
}
```