Laboratory 9: Simple Audio Processor [2 Channel Mixer]

1 INTRODUCTION

The customer loved the initial prototype of your Audio Processor 3000 so much that they have contracted you to add further functionality. You **shall** modify the present instruction set to allow for the playback and mixing of up to two audio files via the instruction formats below.

Play

bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	CH1	CH0	RPT	NA	NA	NA	NA	NA

RPT
- 0: don't repeat
- 1: repeat

CH0
- 0: don't play channel 0
- 1: play channel 0

CH1
- 0: don't play channel 1
- 1: play channel 1

Pause

bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	1	NA	NA	NA	NA	NA	NA	NA	NA

Pausing will prevent audio from playing but the memory pointer should stay at the same location as a future play command should start right where it left off. Pausing should pause both channels.

Seek

bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	0	CH1	CH0	NA	SK4	SK3	SK2	SK1	SK0

SK[4:0]

Memory size per channel is 32768 x 16 bits which requires 15 bits to address. The 5 bits of the seek instruction are prepended to trailing zeros to arrive at an exact memory address to seek to. For example 10010000 has a seek field of 10000 which is prepended to 0000000000 to arrive at 100000000000000 which equates to 16384 or exactly the middle of the ch 0 data memory.

Stop

bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	1	NA	NA	NA	NA	NA	NA	NA	NA

Stop is the same as pause however the memory pointer will be reset to zero. A stop command applies to both audio channels.

Your boss has given you the below sample instructions located in the instructions.mif file and corporate wants results ASAP. You **shall** create an embedded system that will increment through the instruction memory every time the user presses a pushbutton.

```
0  : 0010000000;  -- play ch 1 once
1  : 0001000000;  -- play ch 0 once
2  : 1011001000;  -- seek both channels quarter way
3  : 0011100000;  -- play both channels repeating
4  : 1111000000;  -- stop both channels
5  : 0011100000;  -- play both channels repeating
6  : 0100000000;  -- pause both channels
7  : 1001001000;  -- seek ch 0 quarter way
8  : 0011100000;  -- play both channels repeating
```

After talking to a senior design engineer for a while, you learned that you might need a state machine to manage the communication to the data ROM since there are two data files stored in it however there is only a single address port and data out port. Some potential states are described below.

State	Description of what happens in this state
ch_idle	nothing: proceed to ch0_da on audio_enable pulse high
ch0_da	data address = ch0 address
ch0_audio	ch0 audio = the audio returned from the data ROM
ch1_da	data address = ch1 address
ch1_audio	ch1 audio = the audio returned from the data ROM

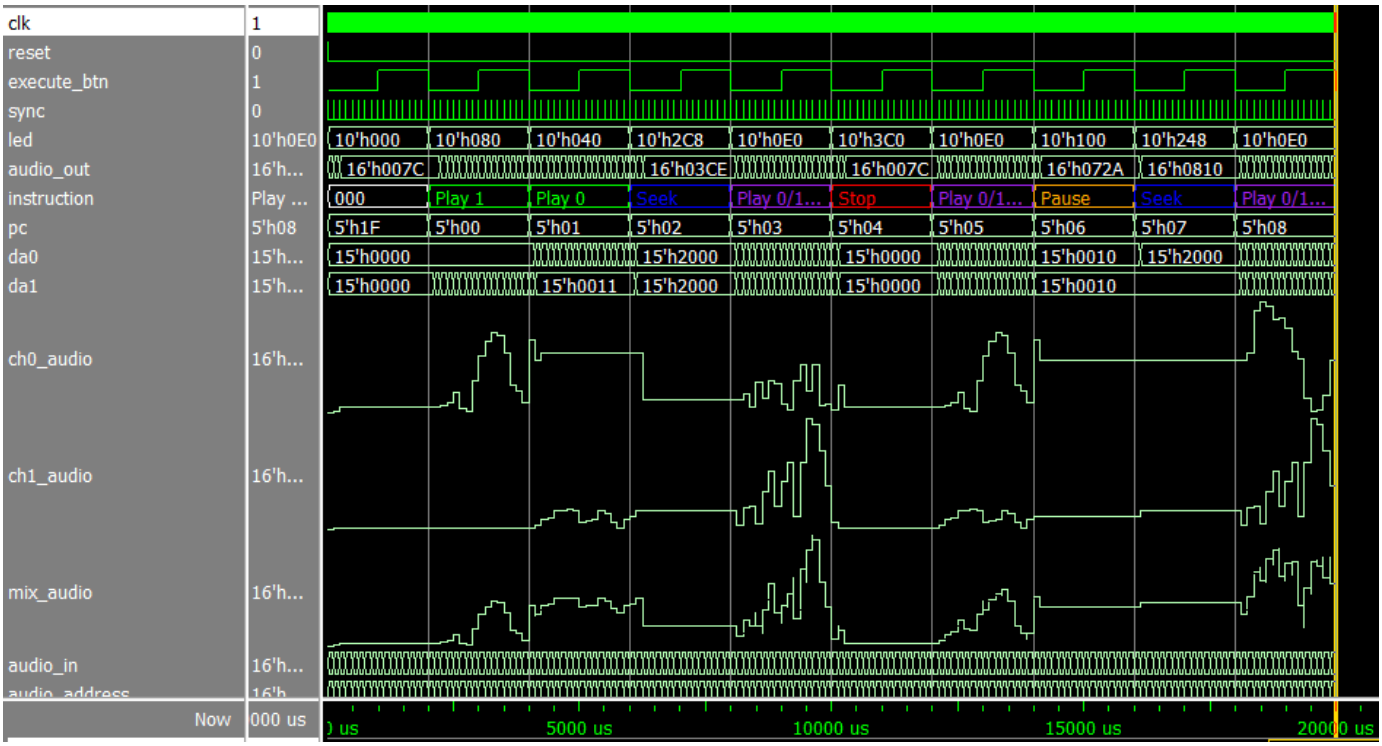
This state machine simply coordinates the access to the data ROM. Think of the state machine as coordinating a multilane highway as it goes down to a single lane (ROM). Only one car can use the single lane road at a time so there has to be a time based coordination system. The basic concept is to start in idle and then to fetch an instruction from the instruction memory on a user key press. Make sure to include a rising edge synchronizer on the key press signal. A fetch will increment the program counter [PC] which will ultimately pull a new instruction from memory. Hint: the PC is another name for the instruction memory address. Once a new instruction has arrived you must decode its opcode to determine what type of instruction it is and also if it is a valid instruction. For some reason the customer does not want to allow one to seek to the very beginning of the memory. An instruction with a decode error will simply be ignored and the system will transition to the idle state.

Your boss is also kind of a control freak and has demanded that your processor have the below entity declaration. Also it was decided that you shall output the instruction byte to the red LEDs.

```
entity audio_processor_3000 is
  port(
    clk           : in std_logic;
    reset         : in std_logic;
    execute_btn   : in std_logic;
    sync          : in std_logic;
    led           : out std_logic_vector(9 downto 0);
    audio_out     : out std_logic_vector(15 downto 0)
  );
end audio_processor_3000;
```

2 SIMULATION

Create the below simulation. Note that the test bench has been given to you in the baseline code. You will need to add an instruction radix which is also posted on myCourses. Note the long duration of this sim and the fact that the audio channels are viewed in the analog format.



3 DELIVERABLES

To receive full credit for this lab one must hand in the below items no later than 168 hrs [7 days] after the start of one's lab session. Signoffs can be obtained after the due date as long as the time stamp of the code is from before the deadline.

- ☐ Hard copy of this document with the block diagram stabled to it. **[Block diagram not due before class]**

4 SIGNOFFS

Category	Initials	Date	Points
Block Diagram			/20
Simulation			/30
Demonstration			/50
Final Grade			/100