CPET 343 Hardware Description Language

Laboratory 9: Simple Audio Processor

1 INTRODUCTION

Congratulations, you have been selected as the lead developer for the new DJRoomba3000 product line! Your first task is to create an embedded VHDL based audio processor that can process the below 8 bit instruction set:

Play

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	RPT	NA	NA	NA	NA	NA

RPT

0: don't repeat

1: repeat

Pause

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	1	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.

Seek

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	0	NA	SK4	SK3	SK2	SK1	SK0

SK[4:0]

Memory size is 16384 x 16 bits which requires 14 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 000000000 to arrive at 10000000000000 which equates to 8192 or exactly the middle of the data memory.

Stop

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

Stop is the same as pause however the memory pointer will be reset to zero.

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

```
0  : 00000000;  -- play once
1  : 00100000;  -- play repeating
2  : 11000000;  -- stop
3  : 00100000;  -- play repeating
4  : 01000000;  -- pause
5  : 00000000;  -- play once
6  : 10010000;  -- seek half way
7  : 00000000;  -- play once
8  : 11000000;  -- stop
9  : 00000000;  -- play once
```

After doing some processor research you discover that you will need a state machine with the following states:

idle, fetch, decode, execute, decode_error

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 report the instruction to the LEDs and the system will transition to the idle state.

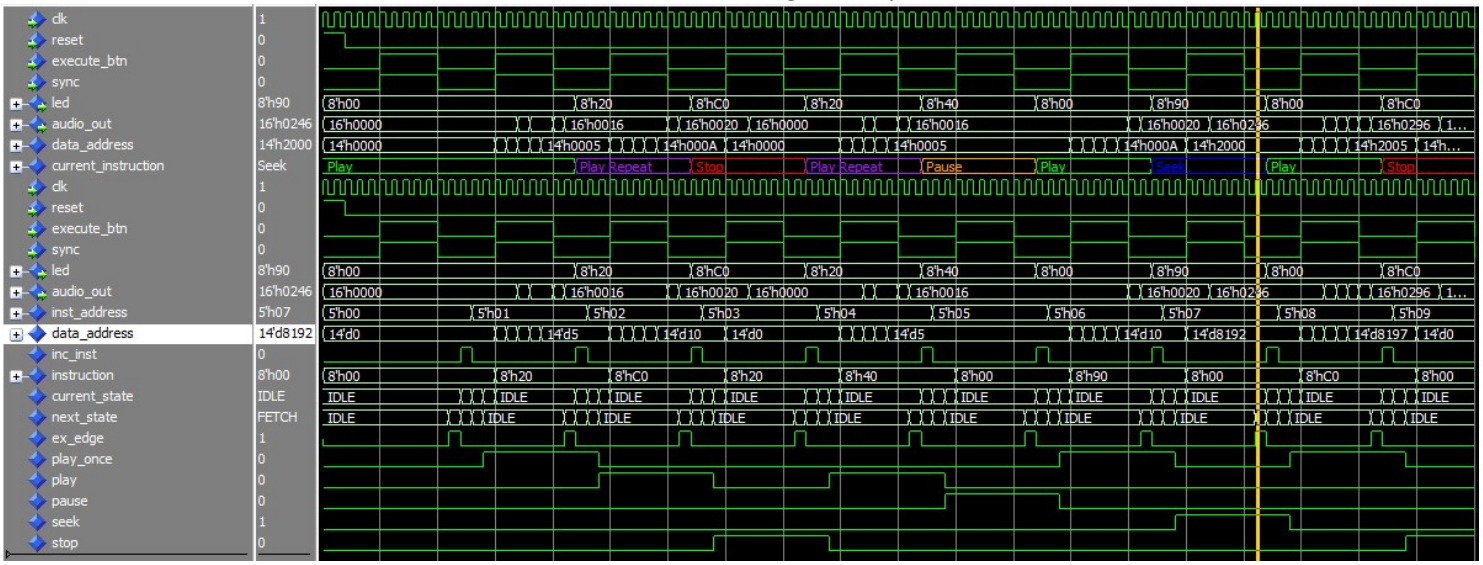
Your boss is also kind of a control freak and has demanded that your processor have the below entity declaration. Also it was decided that you shall output the instruction byte to the red LEDs.

```
entity dj_roomba_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(7 downto 0);
    audio_out     : out std_logic_vector(15 downto 0)
  );
end dj_roomba_3000;
```

But the good news is that you talked to Bob in the lunch room and he already had some code that he worked on for DJRoomba2000 that he gave you as a starting point. His code instantiates a data memory and continuously reads from it and sends the data to the audio codec. All you have to do is compile it via the batch file. Check out the lab8_baseline.zip on myCourses. The code has a top level with an audio codec and a stub for your audio processor in it. **You do not need to modify top.vhd for this lab, but rather use the DJ_ROOMBA_300 file as your sandbox.**

2 SIMULATION

Create the below simulation. Note that the test bench has been given to you.



3 DELIVERABLES

To receive full credit for this lab you must submit all of the following documents to the appropriate MyCourses dropbox.

- □ Completed copy of this document.
- □ Block diagram and state machine.
- □ Snip of Memory Editor Modification using incorrect decode instruction.

Signoffs may be obtained after the due date as long as the time stamp of the code is from before the deadline. Any late signoffs will receive the following deductions:

-10 % if completed within one week of original due date.

-25% if completed within two weeks of original due date.

- Any submission after two weeks will be graded as a zero. Submit work for partial credit at 2 week due date.

4 SIGNOFFS [Lab 9] NAME:

Category	Initials	Date	Points
Simulation			/40
Demonstration			/40
In-Memory Editor Demonstration			/20
Final Grade			/100