Digital Circuit Lab — Lab 3

錄放音機

(Audio Recorder)

Team: 06

Member: 劉容均 b05901084、吳映葦 b05901100、鄧宇凡 b05901183

Github: https://github.com/ywwu928/DClab

- **Introduction**

In this lab, we are going to implement an audio recorder that is capable of record, play, pause and stop. Beside these basic functions, the audio recorder should be able to play at different speed as well as provide a user-friendly interface. In addition, we will let the red LED lights indicate the progress of recording or playing and the green LED lights indicate the volume as some extra bonus.

- **Hardware**: Altera DE2-115 FPGA board
- **Software**: Quartus II
- **HDL (Hardware Description Language)**: System Verilog

Part I: User Manual

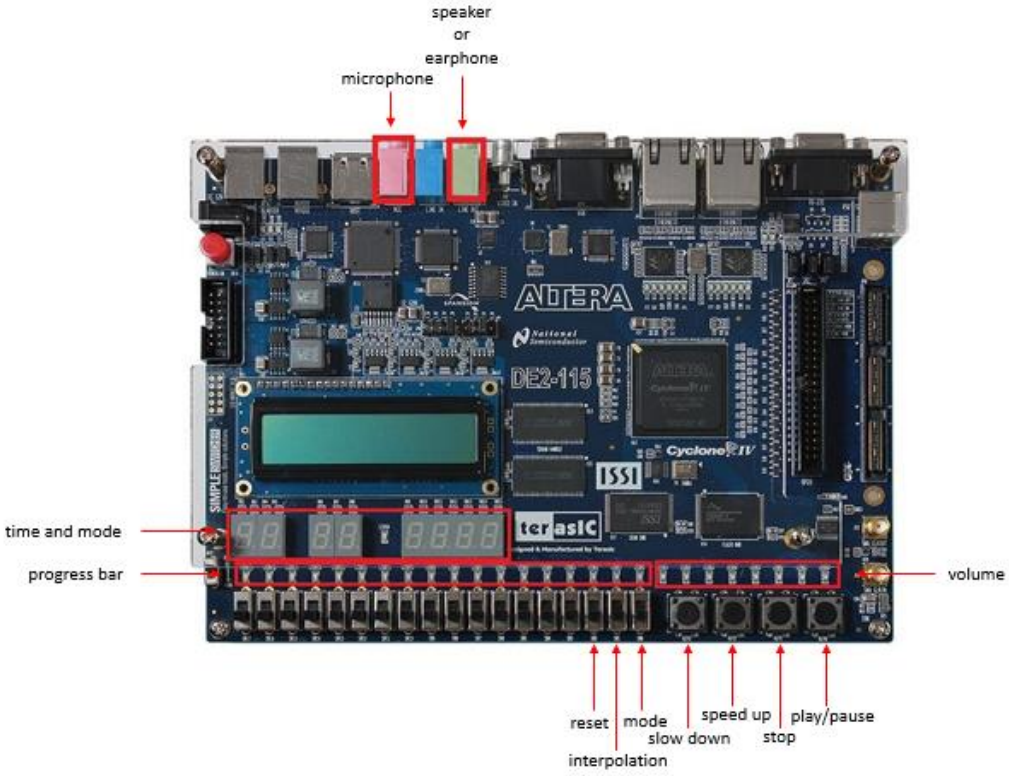
- **Specification**
  - Sampling Rate: 32 kHz
  - Sampling Length: 16 bits
  - Sound Reproduction: Monaural
  - Longest Recording Time: 32 seconds
- **Usage**
  - Connect your microphone and speaker (or earphone) to the FPGA board.
  - Record
    1. Switch SW[0] downward to the RECORD mode.
    2. If the audio recorder is in the RECORD mode, LED17 will be on.
    3. Press KEY[0] to start recording.
    4. You may press KEY[0] multiple times during recording to pause and press again to continue.
    5. When recording, the red LED lights will act as a progress bar to indicate how long you have recorded out of 32 seconds and the green LED lights will show your relative volume.

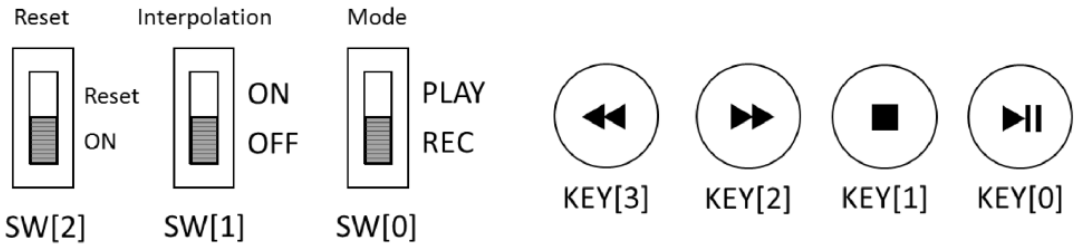
6. When your recording time reaches the maximum 32 seconds, LED15 will be on.
7. To stop the recording process, press KEY[0] first to pause and then press KEY[1] to stop.

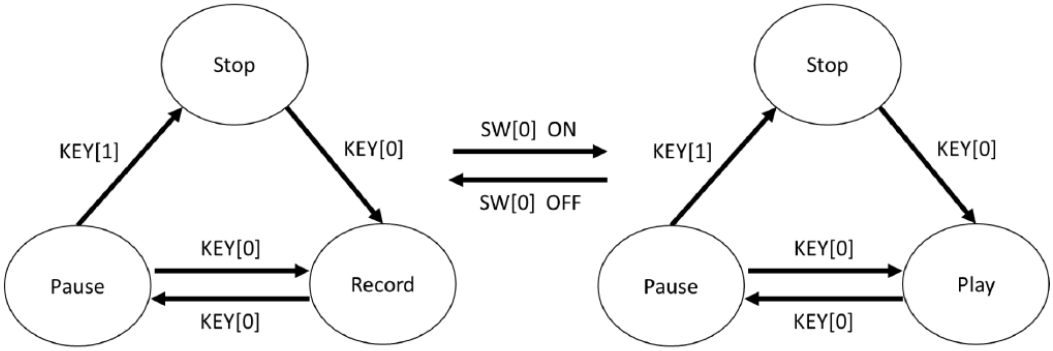
- Play

1. Switch SW[0] upward to the PLAY mode.
2. If the audio recorder is in the PLAY mode, LED16 will be on.
3. Press KEY[0] to start playing the recorded audio.
4. You may press KEY[0] multiple times during playing to pause and press again to continue.
5. When the playing process is paused, you can speed up the audio by pressing KEY[2] or slow down by pressing KEY[3]. The audio recorder supports 2x up to 8x and 1/2x down to 1/8x speed.
6. In slow-forward replaying, you can enable linear interpolation by switching SW[1] upward to ON.
7. When recording, the red LED lights will act as a progress bar to indicate how long you have played out of 32 seconds.
8. To stop the playing process, press KEY[0] first to pause and press KEY[1] to stop.

- **Illustration**





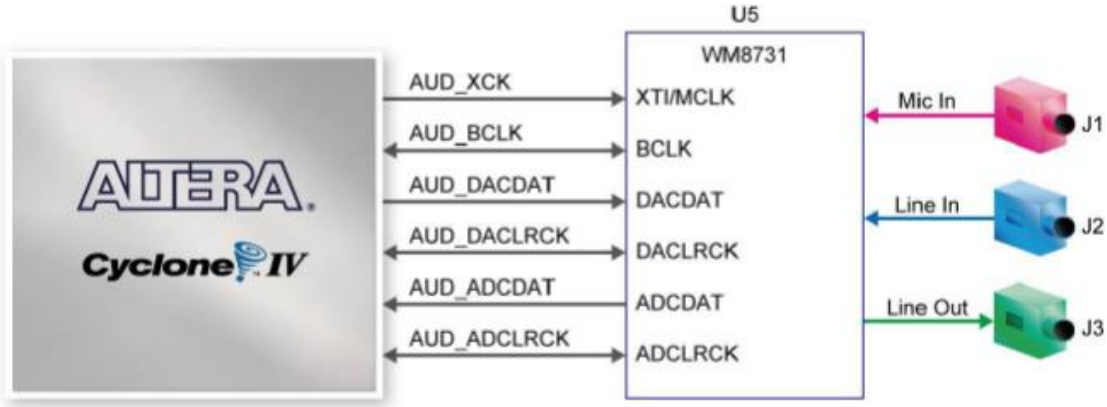


Part II: Tutorial

- **Introduction of Protocols and Tools**

  - WM8731 Audio CODEC

  ➢ Schematic Diagram:



  ➢ Usage:

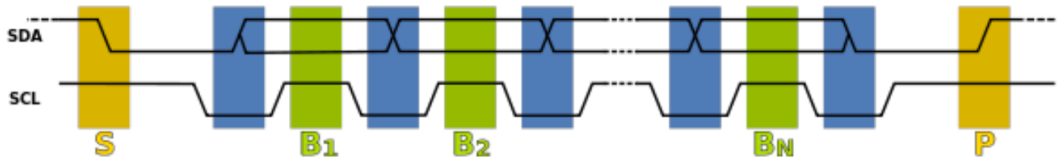
  First, we need to set up the WM8731 audio CODEC by changing its register map, and to do so, we must pass configuration to it with the I^2C protocol. The initialization configuration for this lab are as follows.

Left Line In	000_0000_0_1001_0111
Right Line In	000_0001_0_1001_0111
Left Headphone Out	000_0010_0_0111_1001
Right Headphone Out	000_0011_0_0111_1001
Analogue Audio Path Control	000_0100_0_0001_0101
Digital Audio Path Control	000_0101_0_0000_0000
Power Down Control	000_0110_0_0000_0000
Digital Audio Interface Format	000_0111_0_0100_0010
Sampling Control	000_1000_0_0001_1001
Active Control	000_1001_0_0000_0001

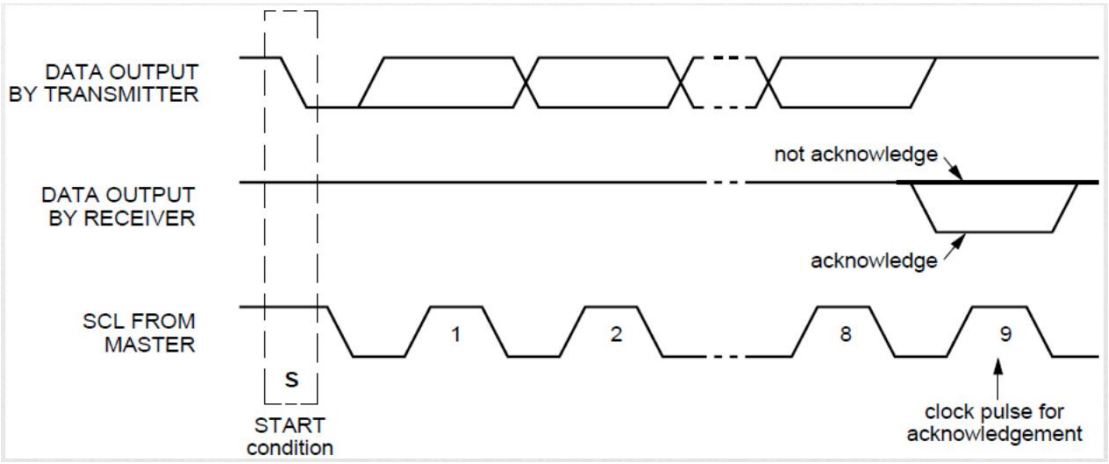
After initializing, WM8731 will be activated as DAC and ADC between audio signal and the FPGA. The sample rate, or the frequency of DACLRCK/ ADCLRCK, is set to 32kHz. Each DACLRCK/ADCLRCK clock cycle contains one sample of the audio signal, which is 16 bits long in our case. The serial data transmission format follows the rules under I^2S mode.

- I^2C Protocol

The picture below is a typical waveform under I^2C protocol, and we shall use it as an example to explain how I^2C protocol works.

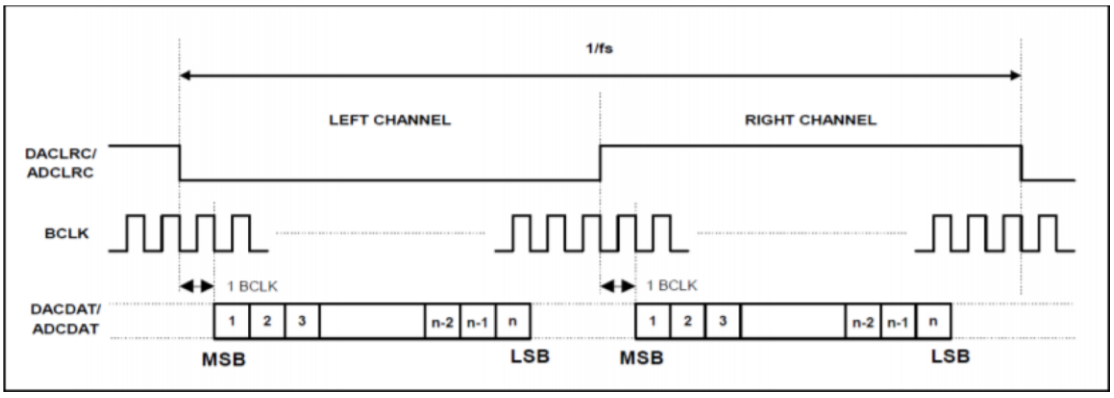


1. The two signals of I^2C protocol, namely SDA and SCL, are preset to high.
2. To start transmission, SDA is pulled low while SCL is high. (refer to the yellow region S)
3. To stop transmission, SDA is pulled high while SCL is high. (refer to the yellow region P)
4. During transmission, SDA sets the data bit to be transferred when SCL is low. (refer to the blue region) While SCL is high, SDA remains unchanged and the data bit is transmitted. (refer to the green region)
5. After transmitting every 8 bits is one direction, an “acknowledgement” bit ACK is transmitted in the opposite direction. If ACK is high, it indicates that the transmission failed; on the other hand, if ACK is low, the transmission is successful. (refer to the figure below)



- I²S Protocol

I²S mode is one of the audio interfaces offered by WM8731. In a single DACLRCK/ADCLRCK clock, DACLRC/ADCLRC low means the left channel while high means the right channel. When DACLRCK/ADCLRCK changes value, DACDAT/ADCDAT will start to record data after one BCLK clock. The number of bits recorded is 16 in our case, and transmission starts from the MSB to the LSB.



- SRAM Communication:

The SRAM on DE2_115 has roughly 2MB memory capacity and is capable of storing 1024K words of 16 bits.

There are seven signals associated with the SRAM:

- SRAM_ADDR[19:0]:
  The read/write address of the SRAM.
- SRAM_DQ[15:0]:
  The 16-bit data to be read or stored in the SRAM.

※ Note that this is a bidirectional inout port, meaning that one must beware of the control of this signal to avoid multiple driver of signal.

- SRAM_OE SRAM:
  Output Enable
- SRAM_WE SRAM:

Write Enable

- SRAM_CE SRAM: Chip Enable
- SRAM_LB SRAM: Lower Byte Control
- SRAM_UB SRAM: Upper Byte Control

The control signals of the read/write operation are listed below:

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O PIN I/O0-I/O7	I/O PIN I/O8-I/O15	V_{DD} Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	I_{CC}
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	I_{CC}
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

Another important point to notice is that although the WM8731 audio CODEC transmits 2's complement formatted binary data in a serial way, SRAM transmits data in a parallel way. In other words, the 16-bit data transmitted from WM8731 can be written into SRAM in a single clock cycle.

- ALTPLL

ALTPLL is a clock rate conversion tool provided by Quartus II. In this lab, we use PLL to convert the original clock rate from 50 MHz to 12 MHz and 100 kHz. The 12MHz clock is then fed to the AUD_XCK port and the 100kHz clock is used to initialize WM8731 through I^2C protocol. For other modules, we use the clock signals generated by WM8731, which is set to 12 MHz and 32 kHz respectively, as the primary clock signals.

Signal	Type	Description
inclk0	input	native 50 MHz clock signal
c0	output	tuned 12 MHz clock signal
c1	output	tuned 100 kHz clock signal

- Test Bench

Test benches are extremely useful for checking whether your module works or not in the initial stage. After running ncverilog in the workstation, you can view your waveforms on nWave and check if each signal behaves just as you expected.

- SignalTap Logic Analyzer

After programming all your modules onto the FPGA board, Quartus II provides a very useful tool to monitor your signal waveforms – SignalTap. It provides instant waveforms and thus is very helpful when debugging run-time errors.

1. Go to File → New → choose SignalTap II Logic Analyzer File → click OK.
2. At the “Setup” section you can select signals you want to monitor, and set the basic clock at “Signal Configuration”.
3. Set the trigger condition, which is basically when to start monitoring the signals you have selected.
4. Save your settings as xxx.stp.
5. Go back to your project and go to Assignments → Settings → SignalTap II Logic Analyzer and specify the stp file.
6. Compile your design.

※ SignalTap basically insert probes in your circuits to monitor the signals specified by the user, so we have to recompile the design to activate them.

7. Program the sof file to your FPGA.
8. Open the stp file once again and switch to the “Data” section.
9. Go to Processing → Run Analysis.
10. Satisfy the trigger condition and start to monitor your signals.

- **System Verilog Files**

- Debounce.sv:

Due to the instability of buttons on FPGA board, Debounce module receives a press-down signal from buttons on FPGA board, and provides a much more stable signal as output for the module DE2_115.

- I2Cinitialize.sv:

Initialize the WM8731 audio CODEC by the I^2C protocol.

- I2Csender.sv:

Transmit data to the WM8731 audio CODEC using the I^2C protocol.

- LED.sv:

Control LED17 to be on when in the RECORD mode and LED16 to be on when in the PLAY mode. Let the other red LED lights act as a progress bar and LED15 will be on upon reaching the maximum recording time 32 seconds. Make green LED lights indicate the volume while recording.

- Para2Seri.sv:

Convert data bits from parallel in to serial out.

- Seri2Para.sv:

Convert data bits from serial in to parallel out.

- SevenHexDecoder.sv:

Given the SRAM address as input, output the current recording or playing time and show the number on the seven-segment display.

- **SevenHexDecoder_State.sv**:

Determine whether you are currently in the state of recording, playing, pause or stop and show the status on the seven-segment display. Also, when changing the speed during PLAY mode, display the speed on the seven-segment display.

- **sramReader.sv**:

Read data bits from SRAM and output the current address.

- **sramWriter.sv**:

Write data bits into SRAM and output the current address.

- **top.sv**:

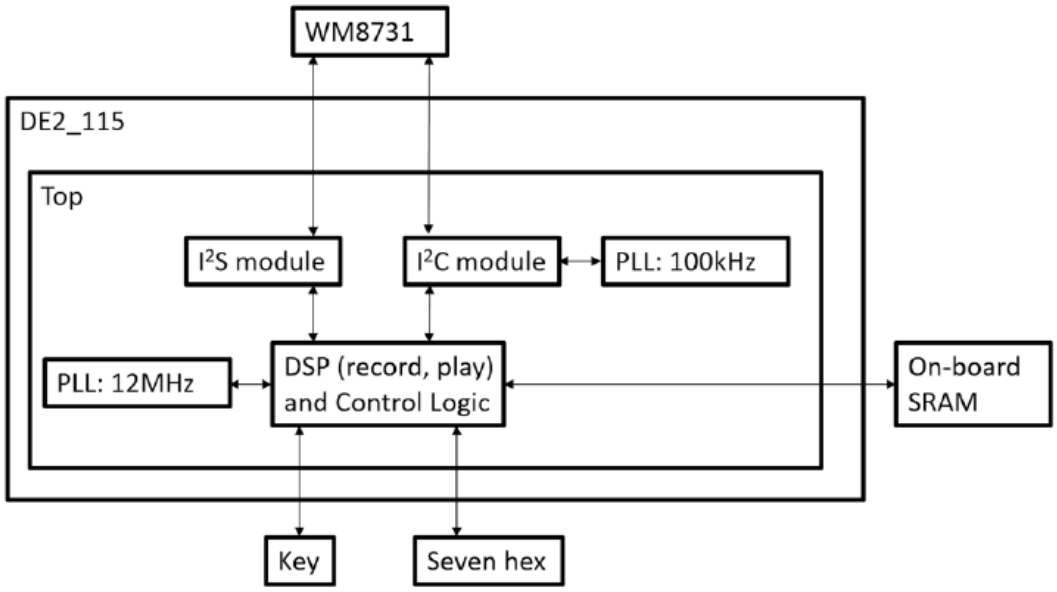
Control all the above modules appropriately to achieve our desired function.

- **DE2_115.sv**:

Uses PLL file to convert clock frequencies and acts as an interface between SystemVerilog codes and FPGA board.

- **Design**

- **Overall Structure:**



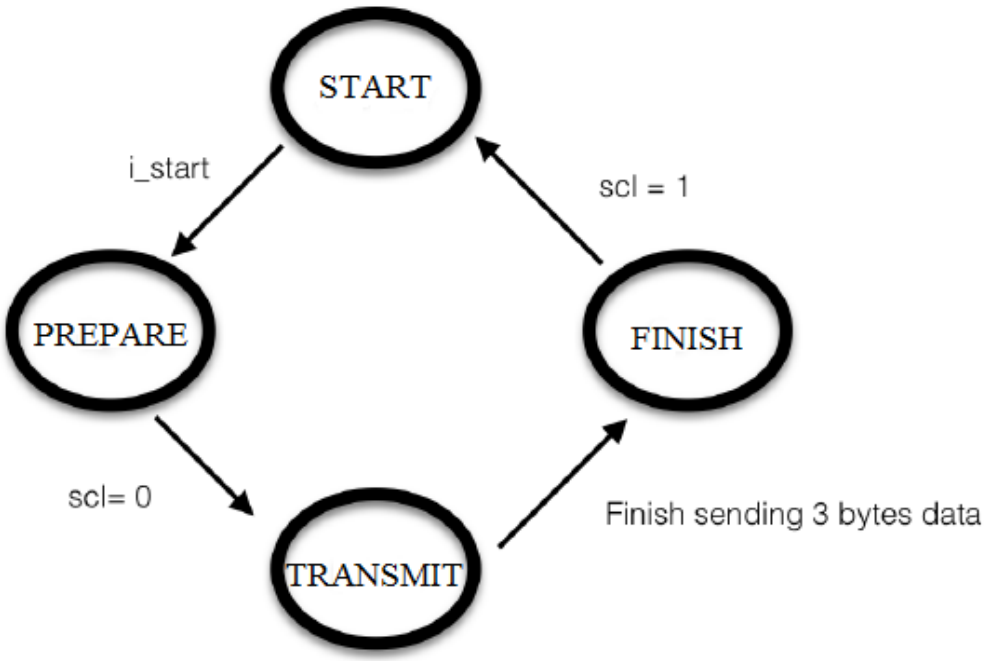
- **I2Csender:**

►**Principle**

In practice, each data transfer would transmit 24 bits of data, which contains three parts. We take the recommended value of Active Control for example.

1. Hardware location of WM8731(leftmost 7 bits): 0011010
2. Location of registers in WM8731(8th ~ 14th bits from the left): 00001001
3. Operation data for register: 0_0000_0001, for example.

► **Finite State Machine**



- START:
  When i_start is high, the module would read the input data (i_data_w = input), and initialize all the counters as well as output data to 0. After initialization, the state would be changed to PREPARE.
- PREPARE:
  When SCL is high, we would set SCL to low state, set output bit to the left most bit of the input data read in START, and shift one bit left of the input data (i_data_w = i_data_r << 1). After doing so, the next state would jump to TRANSMIT.
- TRANSMIT:
  According to the protocol, each time before sending 1 byte data, SCL should be changed from high to low. Therefore, we set SDL = 1 at the first clock cycle and SDL = 0 at the second clock cycle. After doing this, we will start to send the data bit by bit just like what we did in PREPARE (assign output data and shift left 1 bit of input data).
  Upon finishing sending one byte data, we would set the output bit to the ACK signal (1'bz). In practice, we have another signal 'ack' which is set to 1 after sending 8 bit data or otherwise remaining 0 to control the above behavior. For the purpose of depending whether to receive acknowledgement bit or not, we wrote assign o_sda = ack_r ? 1'bz : sda_r.
  After sending 3 bytes data, the next state would be FINISH.

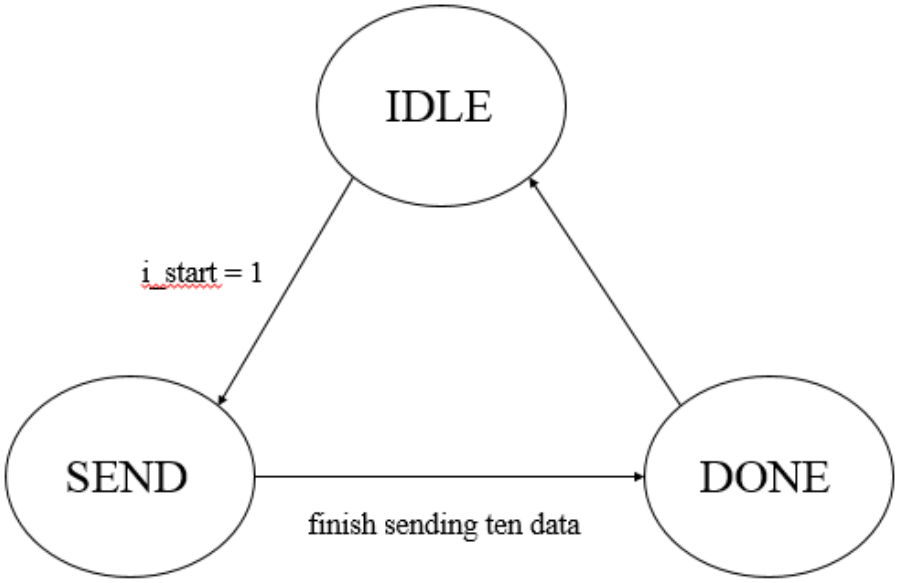
- FINISH:
  If SCL is high, we set both output bit and finish signal to high and the state would jump to START.

- I2Cinitialize:

►Principle

In our design, we have to initialize 10 different registers, so we need to use I2Csender.sv 10 times to send 24 bits data each time.

►Finite State Machine

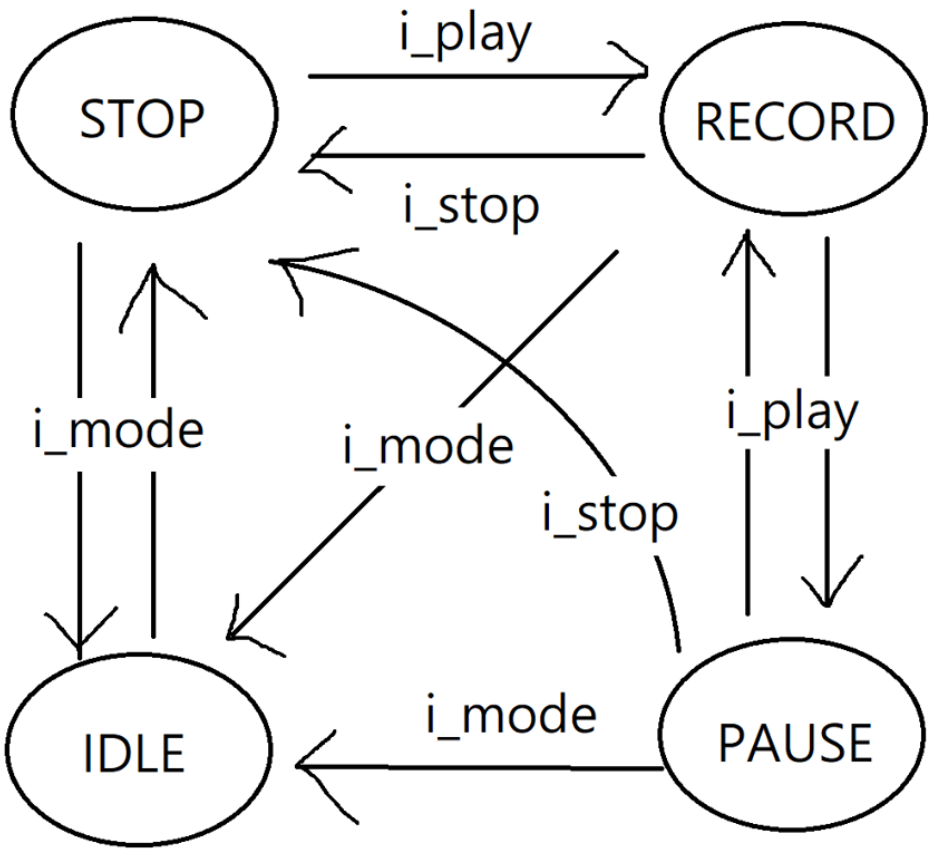


- Recorder:

►Principle

In record mode, we only record the signals from left channel. In each ADCLRCK clock cycle, we receive a stream of 16-bit serial data at the rate of BCLK from WM8731 and transform it to parallel signal. Note that there's a blank BCLK clock cycle after the edge of ADCLRCK. Then we enable the write signal of SRAM to write the data into SRAM and add the address by 1. When the machine runs at modes other than record mode, the output SRAM_DQ should be set to high impedance to avoid conflict when reading data from SRAM. The write enable signal should also be disabled.

►**Finite State Machine**



RECORD MODE

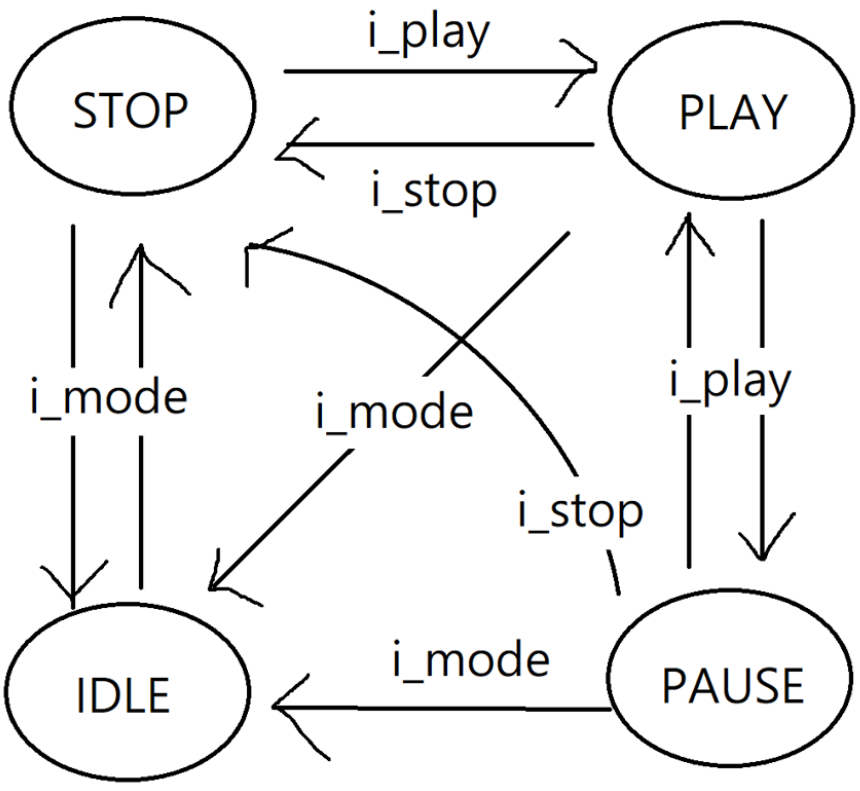
- Player:

►**Principle**

In the play mode, we read the data from SRAM and pass it to the WM8731 by I2S interface. Then WM8731 will convert the digital data to analog signal and output it to audio devices (speaker or earphone). The replay process is pretty straightforward, just simply read the 16-bit data from the SRAM for each address and pass it to the audio chip bit by bit. Note that we pass the same data to the left and right channel since we only records the audio from the left channel. To fast-forward the audio, say, play at 4x speed, we can add the address by 4 instead of 1 for each LRCK clock cycle. Slow-forward is the trickiest part. Assume that we want to play at 1/2x speed. One simple way is to hold each data for two cycles. For even more precise results, we can do linear interpolation. However, the

calculation requires division and multiplication, which are expensive from the viewpoint of hardware, especially division. There are a few ways to approximate the results without using expensive calculations. For instance, we can calculate the step of each cycle $s = \frac{b-a}{n}$ and add the current data by s to avoid multiplication. On the other hand, we can multiply the two data by the ratio and calculate their sum, for instance a×0.25+b×0.75. The ratio is represented as a binary floating point number. By doing it this way, we can avoid division.

►Finite State Machine



PLAY MODE

- Seven Segment Display:

►Timer

Two seven segment displays represent playing or recording time in seconds. As mentioned above, SRAM stores audio digital signals at address range from 0 to 220, we therefore split 0 to 220 into 32 intervals (since the audio recorder records for at most 32 seconds), and by identifying the current reading/writing address

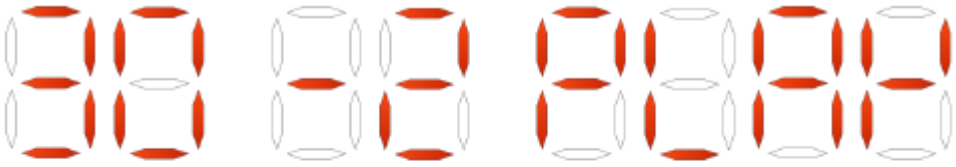
interval of SRAM, we are able to show the correct timer of audio recorder by seven segment display. For example, when SRAM manipulating data at address 0001_1000_0000_0000_0001, seven segment display will show 2 digits number ’04’.

※ Note that we are able to check only most significant 5 bits to identify the address interval.

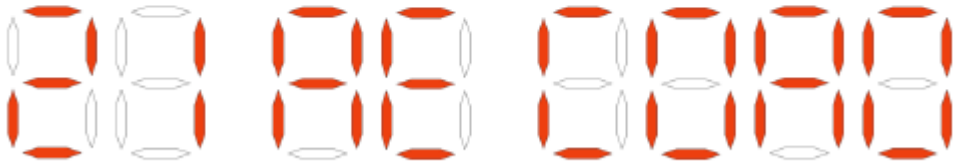
►State

For users to know the current state of the recorder, we utilize the rest of seven segment displays to show the current state of the machine. There are 6 states in total, including init, idle, record, stop, play, and pause. Furthermore, since user may play the recorder at different speeds, we also show the playing speed while the recorder is playing.

►Illustration



Playing at Speed 1/2 to 30 seconds



Recording to 21 seconds