Introduction to our FGPA Board

Over the past month, Zechariah Gerard Tan Jia Le & Xie Yuheng, have been working on programming the logic into our FPGA Board shown in Figure 1 in order to allow it to perform a multitude of functions.

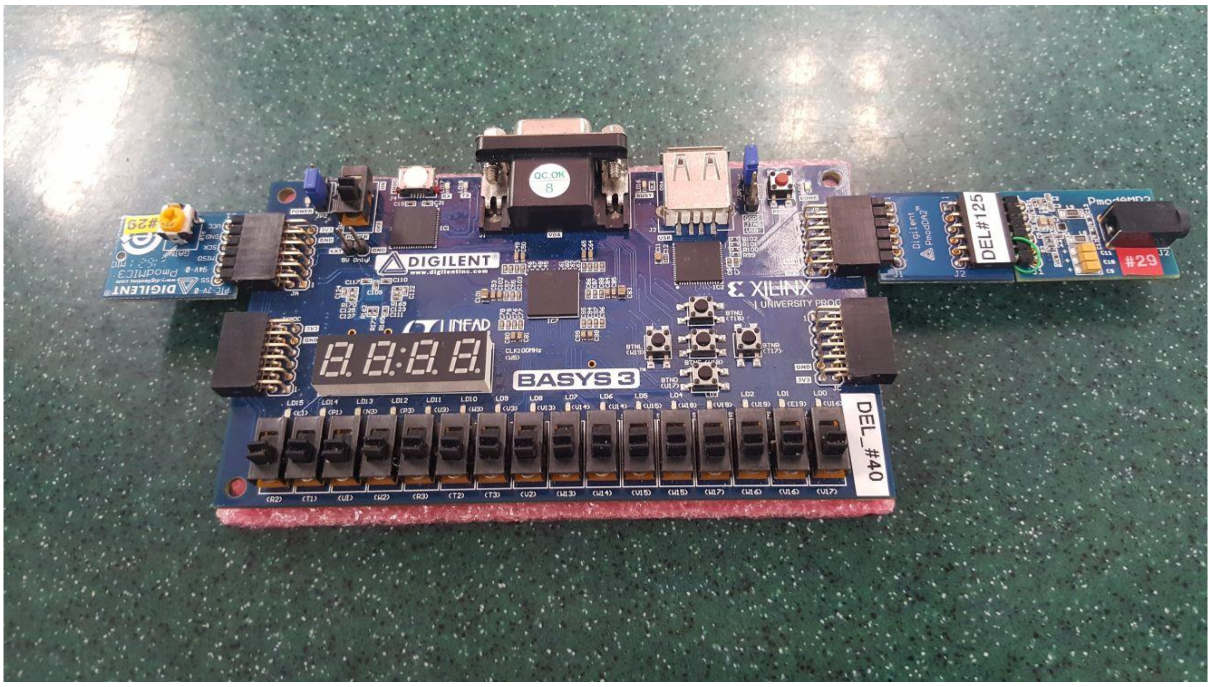
Figure 1 – FPGA Board with its relevant accessories attached

The FPGA Board consists of the *Basys3* board itself, as well as its additional components *PmodMIC3*, *PmodDA2* & *PmodAMP2*, which allows it to perform a multitude of functions as follows:

- Real-time MIC to audio playback
- Volume Indicator
- Musical Instrument
- Delayed MIC to audio playback
- Playing a simple sound repeatedly
- Extra Feature: Playback of 4 preloaded songs

User Guide of Features

Real-time MIC to audio playback
This feature is enabled by default when all the FPGA switches are in the off position i.e. as shown in Figure 1.

This feature, already provided to us, plays back whatever noise is picked up by the microphone in *PmodMIC3* accessory through whatever speaker is connected to the *PmodAMP2* accessory without any delay.

The MIC in *PmodMIC3* accessory picks up noises in the background, and transmits it through the Basys3 board, to the *PmodAMP2* accessory where a speaker can be connected to it to playback the noise picked up.

When the signal is passed from the *PmodMIC3* accessory to the *Basys3* board, the logic programmed into the *Basys3* board converts the audio signal and sends it to the *PmodAMP2* accessory using the module ***SPI.v*** which has been already provided to us.

Volume Indicator
This feature, designed by *Zechariah Gerard Tan Jia Le*, is always enabled when the FPGA Board is on and programmed using our Verilog code.

This feature shows the volume level of the surrounding noise around the MIC on the *PmodMIC3* accessory. Based on the decibel level, the corresponding volume level is shown through the array of the rightmost 12 LEDs (From LED *U3* to LED *U16*) on the *Basys3* board. The more LEDs lighted up, the louder the surrounding noise is.

The MIC in *PmodMIC3* accessory picks up noises in the background, and transmits it to the *Basys3* board, where it is processed and lights up the corresponding LEDs on the *Basys3* Board.

Using the output *speaker_out* provided by the top level module ***AUDIO_FX_TOP.v*** which has been already provided to us, the module ***VolumetoLED.v*** takes in a 2MHz clock as well in order to determine which of the LEDs on the board would light up. The 2MHz clock determines the frequency of update of the LEDs on the board. **It is thus important to note that this feature outputs the volume level provided by other features.** The *speaker_out* input into the module is a 12-bit input which can be represented by its corresponding decimal value. For each of the LEDs on the board in this module, if the corresponding decimal value exceeds a certain threshold value, it would light up. This is shown by the following code:

```
always @ (posedge clk_2M) begin
    temp = 12'b0;
    for(i = 0; i < 12; i = i + 1) begin
        if(volume > 2500 + 133*i) temp[i] = 1;
    end
    LED <= temp;
end
```

For example, the first LED from the right would only light up when temp[0] = 1, which only happens when the corresponding decimal value exceeds 2500. The 11th LED from the right would only light up when temp[10] = 1, which only happens when the corresponding decimal value exceeds 2500 +133(10). These values have been arbitrarily chosen to capture a reasonable range of volume levels, which ignores background noise (0 LEDs lighted up) and caps exceptionally loud noises to the maximum level of 12 LEDs all lighted up.

Musical Instrument

This feature, designed by *Xie Yuheng*, is enabled only when the first switch, labelled *R2*, from the extreme left of the FPGA Board is pushed up.

This feature transforms the FPGA Board to something like a piano, being able to play different combinations of notes ranging from *C4* to *C5* using the 7 switches on the right-hand side of the FPGA Board (From switch *W13* to *V17*). From the right to left are the ascending notes from *C4* to *C5* i.e. switch *W13* plays the note *C4* while switch *V17* plays the note *C5*.

Each of the 7 switches on the right side of the board acts like one key of a piano. By pushing one of the switches up, the board sends an audio signal to the *PmodAMP2* accessory where a speaker can be connected to it to play the corresponding note the switch is. Multiple of these switches can be pushed up to play a musical chord as the audio signal sent by each switch can superimpose on the signals sent by other switches as well.

The musical note output from the board is generated by the logic programmed into the board, which uses the module ***musical_note.v***. Each musical note is generated by means of their own clock with a unique frequency which we found out by doing research of the musical note frequency. As each musical note has its own unique clock, each note has their own module e.g. ***c4_note_clk.v***. The clock(s) of the musical note(s) is the output of ***musical_note.v***.

The notes / clocks produced are then amplified by shifting the 1-bit signal to the most significant bit.

Delayed MIC to audio playback

This feature, designed by *Zechariah Gerard Tan Jia Le*, is enabled only when the second switch, labelled *T1*, from the extreme left of the FPGA Board is pushed up.

This feature plays back whatever noise is picked up by the microphone in *PmodMIC3* accessory through whatever speaker is connected to the *PmodAMP2* accessory with 2 different delay durations. When the switch *V2* is in the off position (pushed down), the delay is 0.25s. When the switch *V2* is pushed up, the delay is 0.5s.

The MIC in *PmodMIC3* accessory picks up noises in the background, and transmits it through the *Basys3* board, to the *PmodAMP2* accessory where a speaker can be connected to it to playback the noise picked up.

When the signal is passed from the *PmodMIC3* accessory to the *Basys3* board, the logic programmed into the *Basys3* board converts the audio signal and sends it to the *PmodAMP2* accessory using the module ***delay_25s.v***. The module takes in the input *MIC_in* provided by the ***SPI.v*** module, a 20kHz clock and the state of the switch *V2*.

By using a 2D array, an array of the instance of the input sound *MIC_in* (which is another array of 12 bits), the current instance of the input sound *MIC_in* is pushed into the start of the array, while all the other instances of previous input sounds are pushed one position down in the 2D array. The last position of the array of input sound instances is overwritten by the previous position, thus whatever instance that occupied this last position is "pushed out" of the array. Figure 2 shows the process:

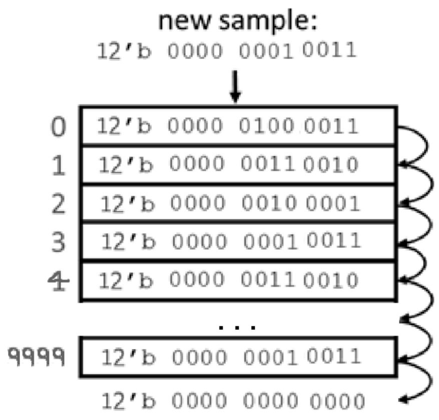

Figure 2 – How the delay feature works by "array pushing"

Depending on the switch *V2*, different instance of *MIC_in* is used as the output of this module. When switch *V2* is off (pushed down), the array position 4999 is used for a delay of 0.25s. When switch V2 is pushed up, the array position 9999 is used for a delay of 0.5s.

Playing a simple sound repeatedly

This feature, designed by *Xie Yuheng*, is enabled only when the middle button, labelled *U18*, is pressed.

This feature makes the FPGA Board continuously play back a simple sound loaded into the board repeatedly until the button is pressed again, where the audio will be paused. The simple sound we used was ***hello20.wav*** which was already provided to us. This audio clip file is converted to an audio signal in the Basys3 board before being sent to the *PmodAMP2* accessory where a speaker can be connected to it can play the loop.

The module ***single_pulse.v*** creates a single pulse using a flip-flop circuit. This single pulse module contains the 2 flip-flop modules ***dff.v*** and generates its own 10Hz clock required for the flip-flop modules. This module generates the output *S* which tells the board when to toggle between the on or off playback state.

Using the following code, the board toggles between playing and pausing the audio loop:

```
always @ (posedge S) begin
    toggle = ~toggle;
end
```

The toggle in this case refers to the playback state toggled by button *U18*.

Next, the module ***playback_module.v*** is the one that gives the audio output through the following code:

```
reg[12:0] address = 0;
dist_mem_gen_0 d1(address, data);
always @ (posedge clock_20k) begin
    address <= ( address == 8192 ) ? 0 : address + 1;
end
```

The 12-Bit data becomes the audio output. We managed to set up ***dist_mem_gen_0.xci*** according to the instructional video provided to us.

<u>Extra Feature: Playback of 4 preloaded songs</u>
This feature, designed by both *Zechariah Gerard Tan Jia Le* & *Xie Yuheng*, is enabled using different switches on the FPGA Board.

This feature makes the board play preloaded songs made by both of us. Pushing up switch *U1* plays the song *River Flows in You* by *Yiruma*. Pushing up switch *W2* plays the song *Odoe to Joy*, the world-famous excerpt from *Beethoven's 9th Symphony*. Pushing up switch *R3* plays a 32.5 second loop of *Canon in* D by *Pachelbel*. Lastly, pushing up switch *T2* plays an 8 second loop of the song *Faded* by *Alan Walker*. The audio output produced by the board is sent to the *PmodAMP2* accessory where a speaker can be connected to it can play the songs.

The module ***songs.v*** gives the overall audio output of all the songs. The sub-module ***song_notes.v*** gives each of the notes in every song its sound.

The music note output is updated every 1 millisecond using the 1 kHz clock. The durations of the songs are hardcoded into the module, and when the duration exceeds the duration specified by the song, duration gets reset to 0 for the song to be looped. Otherwise, it simply adds 1 to the duration which is in milliseconds. In each of the songs, the duration of each note is hardcoded according to the actual song themselves, which we used online piano score sheets as references. The note to be played is a 5-bit which then goes into the module ***song_notes.v***.

In the ***song_notes.v*** module, the 5-bit input determines which of the frequencies to use for the output clock. It does so by hardcoding the wavelength of each of the musical notes from *F3* to *B5*. Letting the wavelength be `value`, the following code allows this module to generate the appropriate frequency:

```
always @ (posedge CLK) begin
        COUNT <= ( COUNT == value ) ? 0 : COUNT + 1;
        out_note <= ( COUNT == 0 ) ? ~out_note : out_note;
end
```

The higher `value` is, the longer it takes for `out_note` to be toggled, thus the longer wavelength and the lower the frequency.

The code for this feature was adapted from the reference website below. The songs we used are different from the one provided by this reference.
https://github.com/z-e-r-0/Songs

Overall Design of our Verilog code

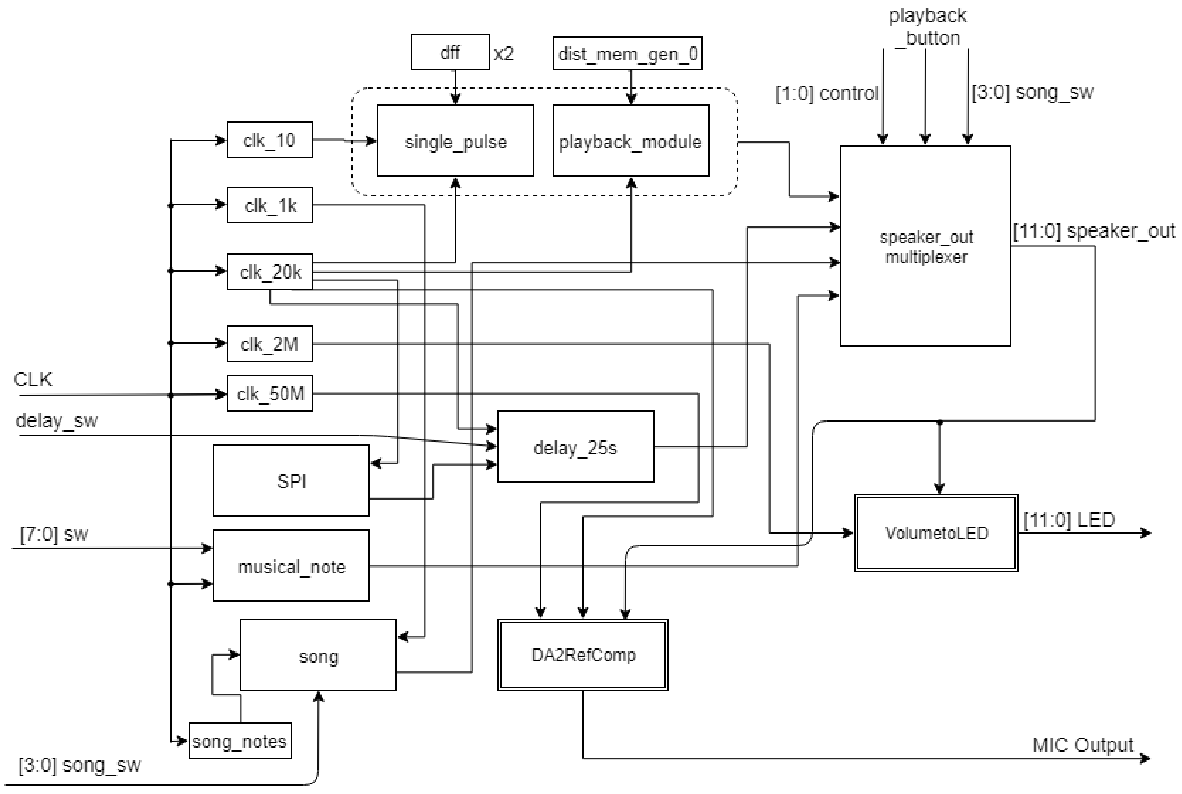

Figure 3 – Overall module design of our Verilog code

In Figure 3, we used the default *CLK* provided by Verilog to create the other clocks, namely *clk_10*, *clk_1k*, *clk_20k*, *clk_2M* & *clk_50M*, using this general code template:

```
module clock_f(input CLK, output reg SLOWCLOCK);
    reg [y:0] COUNT = 0;
    always @ (posedge CLK) begin
        COUNT <= ( COUNT == x ) ? 0 : COUNT + 1;
        SLOWCLOCK <= ( COUNT == 0 ) ? ~SLOWCLOCK : SLOWCLOCK;
    end
endmodule
```

In the above code, for the desired clock frequency `f`, integers `x` and `y` are chosen such that:

$x = (50*10^6) / f$
$2^{y+1} \geq x$

For example, for our 20kHz clock, the values `x = 2500` and `y = 11` were chosen.
For our 50MHz clock, since the values `x = 1` and `y = 0` were chosen to make this clock, then the second line of the code simply becomes:
```
reg COUNT = 0;
```

In addition, with reference to Figure 3:

- *clk_20k* is the input for ***SPI.v***
- *clk_20k*, *MIC_in* (provided by ***SPI.v***) & *delay_sw* are the inputs for ***delay_25s.v***
- *CLK* & *[7:0]sw* are the inputs for ***musical_note.v***
- *clk_1k* & *[3:0]song_sw* are the inputs for ***song.v***
- *clk_20k* & ***dist_mem_gen_0.xci*** are the inputs for ***playback_module.v***
- *clk_10*, *clk_20k* & 2 x ***dff.v*** are the inputs for ***single_pulse.v***
- *clk_2M* & *[11:0]speaker_out* are the inputs for ***VolumetoLED.v***
- *clk_50M*, *clk_20k* & *[11:0]speaker_out* are the inputs for ***DA2RefComp.vhd***

The outputs produced by ***delay_25s.v***, ***musical_note.v***, ***song.v***, ***playback_module.v*** & ***single_pulse.v***, are then fed into the speaker_out multiplexer (by means of the selection statement shown below) and using the inputs *[1:0]control*, *[3:0]song_sw* & *playback_button*, the output *[11:0]speaker_output* is produced.

```
assign speaker_out = control[0] ? swap_bit : (control[1] ? speak2_out :
(song_sw[0] ? amp : (song_sw[1] ? amp : (song_sw[2] ? amp : (song_sw[3] ?
amp :(toggle ? playback_out : MIC_in))))));
```

The output *[11:0]speaker_output* produced is then fed into ***DA2RefComp.vhd*** which produces the MIC Output to the *PmodAMP2* accessory. The output *[11:0]speaker_output* is also fed into ***VolumetoLED.v*** to produce *[11:0]LED* which produces the LED outputs on the *Basys3* board.

Feedback

1) What did you like most / least about the project?

Positives:

- Being able to see your feature work properly is definitely a joy, similar to programming when your code finally is able to solve the problem task.
- This task challenged my thinking on how the code should be designed.
- People doing the same tasks can come together to work on the code.

Negatives:

- The jump in difficulty was too great from lab 2 to lab 3
- This was because we have not even learned the relevant theoretical knowledge (such as CLKs)
- The flow of the module was also very haphazard due to the above, as well as the fact that only **after the lab quiz** were we properly taught the syntax of Verilog, which makes utterly no sense
- There is little way for us to check whether our code can produce an error-free synthesis, implementation & bitstream due to the Vivado not properly checking everything before running these processes. This is especially frustrating when the whole process can take up to 20 minutes or so (as it is in our project)
- There is not enough guidance given for this project in the instruction manual.
- The lab quiz expects too much out of us especially since it involves the 7-segment display.

2) How would you suggest the overall project assignment be improved?

- Reorder the way content is being taught in lectures. Verilog should come relatively early. All coding requires tons of practice and we are not exposed to enough.
- More guidance given in instructions, such as more hint codes, since most people cannot figure out codes.
- Find a better Verilog program, or provide a program that can check for correctness of the overall code.

3) Any other constructive feedback / suggestions are welcome.
Already mentioned in Q2.