Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

G19 Laboratory #5
Report: g19_block5

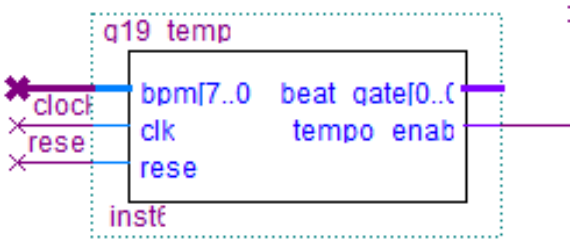
In this laboratory, our team will combine all the different parts of the previous laboratory together in order to build a Music Box. The music box should be able to play two different songs. Our circuit includes the following parts (See Table 1):

Table 1: Parts from all the laboratories

Parts	Laboratory	Functions
g19_tempo	3	• Adjust the beat of the song by changing the bpm • Frequency Divider
g19_Envelope	3	Used to give a right sound wave and beat for the block responsible to decode it and play it.
g19_square_wave	4	• Generate the basic waveform to provide musical notes • Frequency Divider is used
g19_note_timer	4	Control the tune to be played by the music box (16-bit note information)
g19_audio_interface	4	Implements the serial communication to an Audio Codec chip on the board (Provided by Prof. Clark)
g19_decimator	4	Convert the sampling rate of 50 MHz to 48 KHz from the square wave generator (Provided by Prof. Clark)
g19_Finite_State_Machine	5	The overall control of the music box

In the laboratory #3, a timing of the notes is constructed. (See Figure 1) It is called *g19_tempo* and it is measured in Beats Per Minute (BPM). In other words, it controls how fast the beat will be played. A programmable frequency divider is used. A counter, which is a sequential circuit, is created by the MegaWizard and the count sequence of the clock is descending. Once the counter reaches zero, the BPM value, which are stored in a Memory Initialization File (MIF) is fed into the counter and counts down from that value.

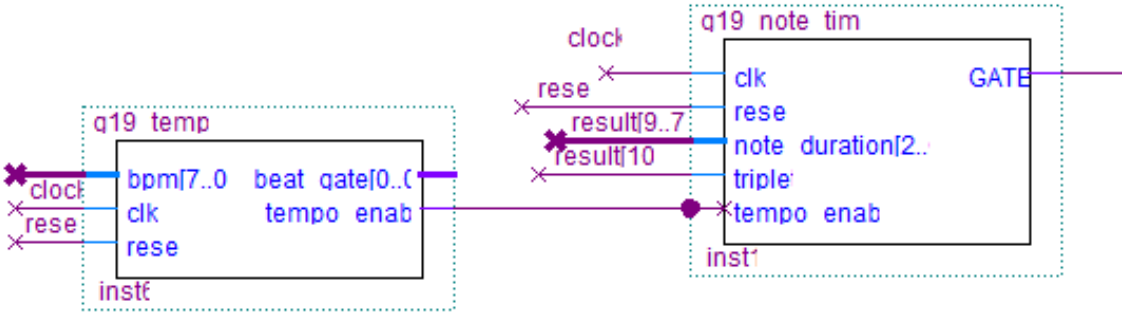
Figure 1: g19_tempo from Lab 3



Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

The bpm is a 8-bit value and generate a tempo_enable signal (a slower frequency) for the g19_note_timer. (See Figure 2)

Figure 2: Connection of g19_tempo and g19_note_timmer



The note timer built in lab4 is used to describe the tune to be played by our music box(triplet, note duration). A 16-bit note information words are stored in a song memory. (See Figure 3) Also, in figure 4, one can see the information contained in the 16-bit word.

Figure 3: The song memory

Figure 4: Note Information

1. (bits 0-3) Note Number
2. (bits 4-6) Octave Number
3. (bits 7-9) Note Duration
4. (bit 10) Triplet
5. (bits 11-14) Loudness
6. (bit 15) End of Song marker

Each note number represents a specific note in the musical notes whereas the octave number shows which octave the note is to be played at. The note duration and the triplet information are used to generate an appropriate note GATE signal (correct high and low period) which match the transition table in the laboratory 4 description. (See Figure 5 and 6) The Loudness

controls the volume of the music box and it occupies the bits 11 to 14. Finally, the End of the Song marker will indicate the end of the song.

Figure 5: Note duration when triplet bit is zero

- ***Note duration => number of tempo pulses per note***
- 0 => 3 (one eighth beat long – equivalent to a thirty-second-note in musical terminology)
- 1 => 6 (one quarter beat long, or a sixteenth-note)
- 2 => 12 (half a beat, or an eighth-note)
- 3 => 24 (one beat, or a quarter-note)
- 4 => 48 (two beats, or a half-note)
- 5 => 96 (four beats, or a whole-note)
- 6 => 192 (eight beats, or two whole notes)
- 7 => 384 (sixteen beats, or four whole notes)

Figure 6: Note duration when triplet bit is one

- ***Note duration => number of tempo pulses per note***
- 0 => 2 (one sixth beat long – equivalent to a dotted sixty-fourth note in musical terminology)
- 1 => 4
- 2 => 8
- 3 => 16
- 4 => 32
- 5 => 64
- 6 => 128
- 7 => 256

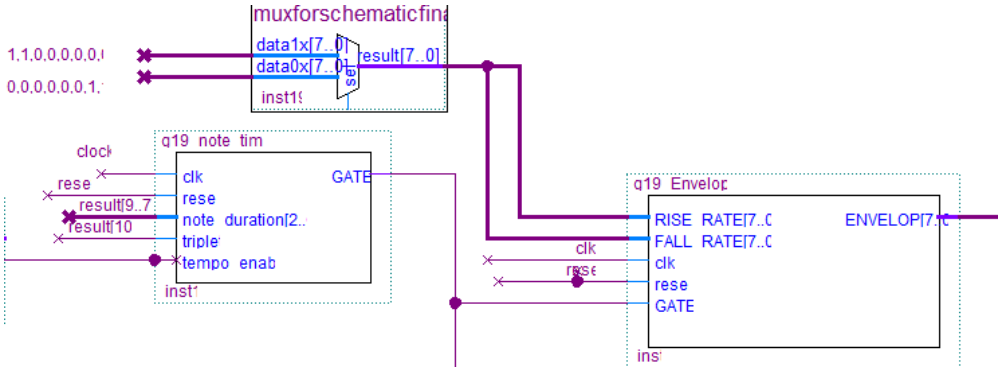
The circuit is implemented by using a single process block counter. The circuit takes in a signal called *tempo_enable* which is generated from the *g19_tempo*. Whenever tempo_enable is high, the value of count is incremented on every clock cycle. Within the note_timer clock, we use process block to implement different operation of the circuit. The *GATE* is set to zero when the count reaches half of the number of the note duration. If the count value reaches the number of note duration minus 1, then the *GATE* is set to 1 again and the count is set to 0.

The *GATE* signal of the g19_note_timer is then used as the input of the *GATE* signal input of g19_Envelope module. When *GATE* is high, the output of the envelope circuit will increase. On the other hand, if the *GATE* is low, the envelope will decrease.

Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

A multiplexer is implemented in order to choose the two different set of 8-bit *RISE_RATE* and *FALL_RATE*, which are set to a constant value of 192 and 3 respectively which will give the song are more “staccato” feel when the rate is high and a more “continuous” feel when the rate is low. (See Figure 7)

Figure 7: The block diagram of the g19_Envelope attached to a MUX and g19_note_timer



The output of the envelope circuit is multiplied to the volume value corresponding to the Loudness bits coming from the song ROM output. (See Figure 8) Loudness is converted to the 23 bit volume input to the square wave generator. (See Figure 9) The conversion can simply be done with a selected signal assignment in VHDL.

Figure 8: Multiplication module with the 23-bit volume and 8-bit output of envelope

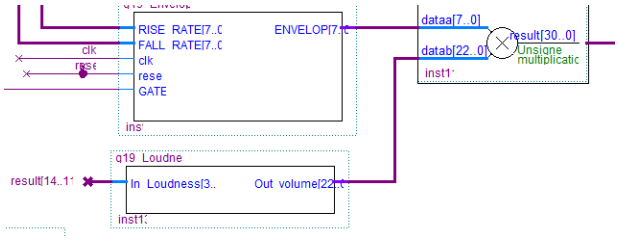


Figure 9: Conversion of Loudness to 23-bit volume

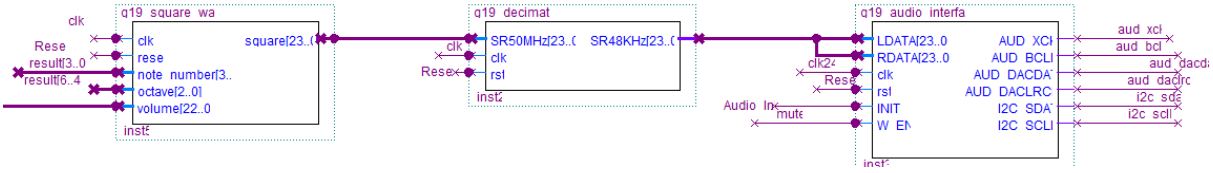
- ***Loudness => volume***
- 0 => 00000000000000011111111
- 1 => 00000000000000111111111
- 2 => 00000000000001111111111
- . . .
- 13 => 00111111111111111111111
- 14 => 01111111111111111111111
- 15 => 11111111111111111111111

Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

The product of the multiplication of the output of envelope and volume module will be feed into the Volume input of the g19_square_wave module. The square wave module is capable generating a basic waveform to provide musical notes. In the *g19_square_wave* circuit, a 4-bit binary value, which represents the note number between 0 to 11, is used to create the period of the pitch, which is a 20-bit value. The *pitch_period* is shifted by an amount octave (3 bits). Consequently, a frequency divider is used to generate a waveform and it is generated with a counter that counts from the pitch_period down to 0. As the counter reaches the value 0, the counter resets.

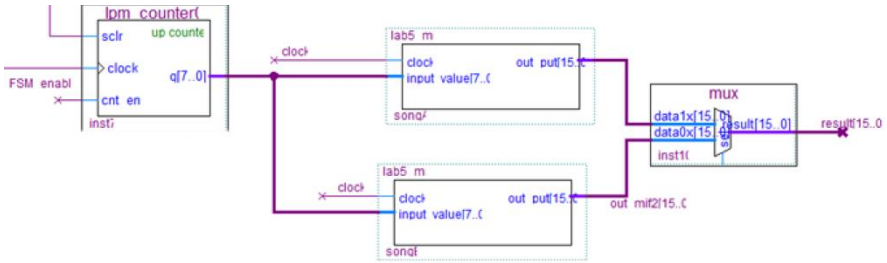
Also, the two audio modules, called it g19_audio_interface and g19_decimator, are provided by Prof. Clark. The audio interface is a module that implements the serial communication to an Audio Codec chip located on the Altera board. On the other hand, concerning the decimator, another codec chip helps to convert the sampling rate of 50 MHz to 48 KHz from the square wave generator. (See Figure 10)

Figure 10: g19_square_wave attached to two audio modules



Also, a counter is added to keep track of which note is being played. It is used to indicate the address for the two songs ROM. Furthermore the enable of the counter is used by the Finite State Machine (FSM) which helps to control play, pause, stop, stop state of the music box. The output of g19_note_timer, which is *GATE*, is connected to the counter in order to determine when the note is completed and then increment the address counter of the song ROM, to go to the next note. (See Figure 11) Both songs ROM are connected to a MUX in order to choose the desired output of one of the songs.

Figure 11: The counter for the songs



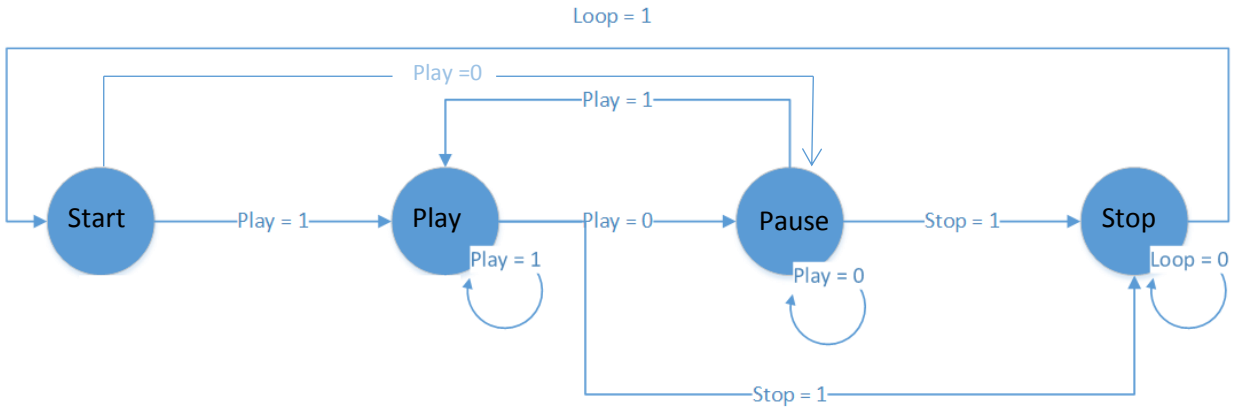
Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

Furthermore, a Finite State Machine is implemented in the music box and it has 4 states, which are Start, Play, Pause and Stop. (See Figure 12) The music box has to play 2 different songs, 256 notes: One of them is called Für Elise and is defined by the file g19_demo_song.mif provided by Prof. Clark and the second song is the theme song of Legend of Zelda[1].
Our FSM has 3 state variables, which are *p*, *stop_bit* and *Loop*. They represent the pause signal, stop and loop signal respectively. For example, if p equals to one from the state Play, it will go to state Play. However if p is zero, it will travel to the Pause state. In table 2, it shows the present and the next state of our FSM. The song will continuously loop itself if *Loop* is equal to 1.

Table 2: Present and Next State

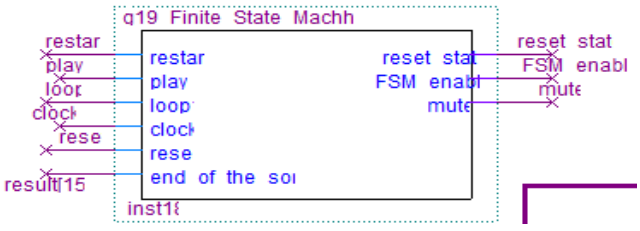
Present State	**Next State**				
	P=1	P=0	Stop_bit=0	Stop_bit=1	Loop = 1
Start	Play	Pause	Don't Care	Don't Care	Don't Care
Play	Play	Pause	Don't Care	Stop	Don't Care
Pause	Play	Pause	Don't Care	Stop	Don't Care
Stop	Don't care	Don't Care	Don't Care	Don't Care	Start

Figure 12: Finite State Machine



A module of the FSM is also created by compiling the VHDL code. (See Figure 13)

Figure 13: Block Diagram of the FSM

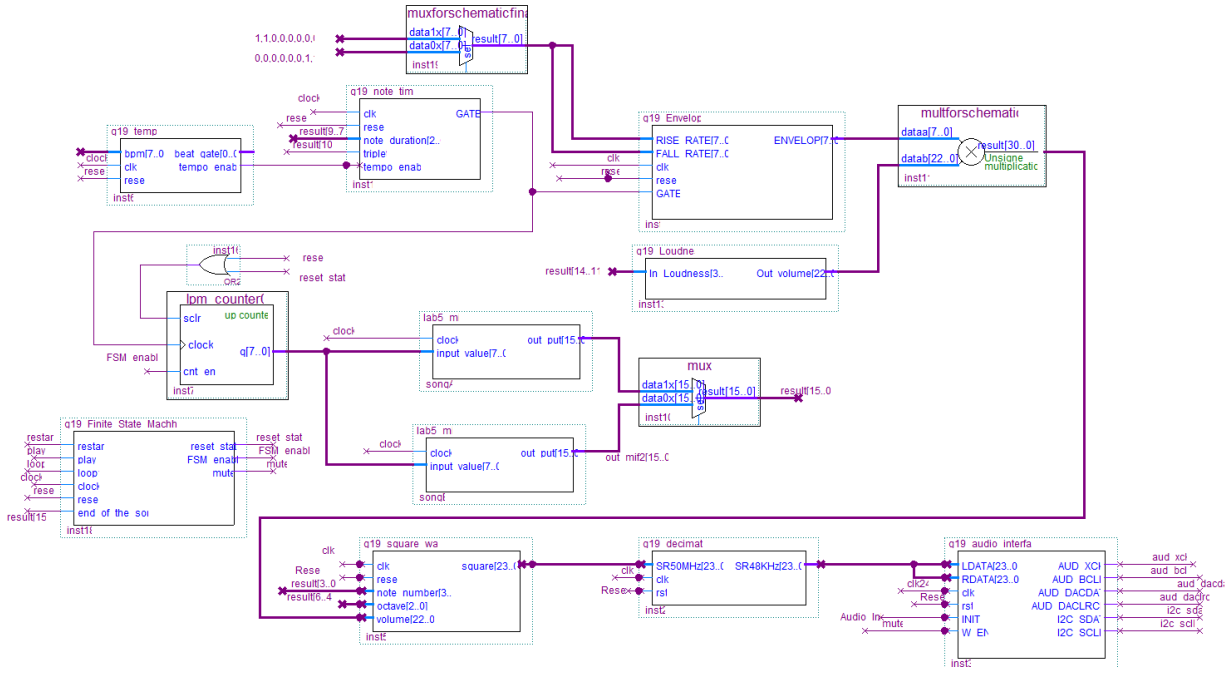


Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

The inputs are the 4 states of the FSM, where the MSB of the 16-bit note information words is used as the Stop State and marks the end of the song. In addition, there is a clock and reset inputs. Concerning the outputs, the *reset_state* is fed into the counter and keeps track of which note is being played. If the *reset_state* is high, it will acts as a synchronous clear and resets the counter. The *FSM_enable* is used to the *count_enable* of the counter shown in Figure 11. Finally, the *mute* signal is fed into the g19_audio_interface.

The overall block diagram of the entire music box is shown in Figure 14.

Figure 14: Entire Block Diagram of the Music Box (g19_block5)



All the states (playback mode, song start, and pause/resume) are controllable by different slide switch of the DE1 Board. The Start indicates the start of the song, the Play state represents the moment when the songs is playing, Pause is a state where the song is paused and Stop state is when the song stops. Note that the Pause state acts as a playback system. (i.e. the song can be paused and resumed in the middle of a song using a slide switch) By toggling the switches, we can also select 2 modes: *One-Shot* and *Continuous Looping*. The display of the beat, play mode and system status is displayed on the LEDs. To do this, our team used what we learned in laboratory #2 and generated the appropriate 7-segment display with VHDL code for displaying “LOOP” for loop mode and “SHOT” for the one shot mode.

Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

Six different slides switches are used to control the variation of the tempo between 46-300 BPM in steps of 4 BPM. The LED should also flash at the BPM rate. Also, one can easily playback the song from the beginning by pressing a *reset* pushbutton.

Last but not least, the speed at which the song is playing is controllable by 2 different envelope switches of rise and fall rates.

The system was tested by verifying the variation of the sound produced by different rise, fall rates and BPM rate. The rise and fall rate will give the song a more "staccato" when both rate is high and a more "continues" feel when both rate is small. The BPM is assigned to 6 switches on the board and will change the tempo/ speed of the song. If the *pause* switch is set to high, the music will stop to play, however, if the same switch is toggled back to its original position, the state will change to the *resume* and the song will continue to play from where it stopped. The *reset* pushbutton can be pressed to restart the whole song. Finally, both of the song can be played properly.

We assign one switch to dedicate to toggle between the loop and shot mode. When it is in the loop mode, the display on the board will display "LOOP" ("SHOT'' will be displayed when it is in the shot mode) and the song will loop itself over and over again until the mode is changed or the state is *Pause*.

There is also one switch to toggle between songs. When this switch toggles, the current song that is playing will be stopped and second song will then be played by reading a different mif file. However the count will be reset so the song will be start from the beginning instead of the middle of the song where the other song left off.

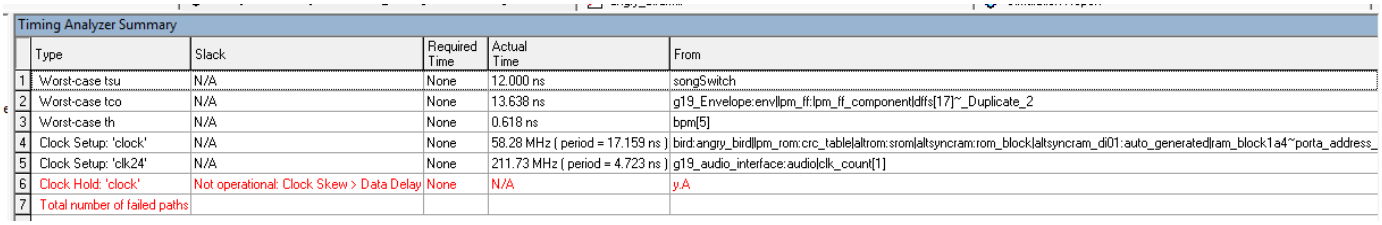
In the Flow Summary, it indicates that we used 1352 transistors for the simulation of the square wave (See Figure 15). The largest propagation delay of the circuit is 13.638 ns. (See Figure 16)

Figure 15: Flow Summary

Flow Status	Successful - Wed Nov 27 13:13:44 2013
Quartus II 64-Bit Version	9.1 Build 350 03/24/2010 SP 2 SJ Full Version
Revision Name	g19_lab4
Top-level Entity Name	lab5_block
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	No
Total logic elements	1,352 / 18,752 (7 %)
Total combinational functions	951 / 18,752 (5 %)
Dedicated logic registers	978 / 18,752 (5 %)
Total registers	978
Total pins	65 / 315 (21 %)
Total virtual pins	0
Total memory bits	13,794 / 239,616 (6 %)
Embedded Multiplier 9-bit elements	4 / 52 (8 %)
Total PLLs	0 / 4 (0 %)

Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

Figure 16: Timing Analyzer



Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From
1	Worst-case tsu	N/A	None	12.000 ns	songSwitch
2	Worst-case tco	N/A	None	13.638 ns	g19_Envelope:env|lpm_ff:lpm_ff_component|dffs[17]~_Duplicate_2
3	Worst-case th	N/A	None	0.618 ns	bpm[5]
4	Clock Setup: 'clock'	N/A	None	58.28 MHz (period = 17.159 ns)	bird:angry_bird|lpm_rom:crc_table|altrom:srom|altsyncram:rom_block|altsyncram_di01:auto_generated|ram_block1a4~porta_address_
5	Clock Setup: 'clk24'	N/A	None	211.73 MHz (period = 4.723 ns)	g19_audio_interface:audio|clk_count[1]
6	Clock Hold: 'clock'	Not operational: Clock Skew > Data Delay	None	N/A	y.A
7	Total number of failed paths				

In conclusion, our team ran into several problems during the design process. Since our team does not have a deep knowledge of music, we had trouble composing our own song. Therefore, we decided to pick our second song from a website and translate it into a MIF file by using Microsoft Excel. Our team had difficulties playing the middle of the song if we decide to switch songs. In other words, if we played the first half of song A, and then decided to switch to song B, we will have to play song A from the beginning instead of playing it from the middle.

Also, our design was limited to play one note at a time. Our team could have programmed our music box such that it can play several songs at a time, and therefore, have more inspiring song. Furthermore, the 7-segment decoder can be designed to display the other states of the Finite State Machine. In our case, depending on the modes, the music will only display *SHOT* or *LOOP*. (*One-Shot* or *Continuous Looping*) The states, Start/Play/Pause/Stop, can be added in the 7-segment LEDs and therefore, the user has a better idea of the states he is in. To wrap things up, we successfully created a music box system by combining all the parts of the laboratory and we have a better knowledge of VHDL and the convenience of using the Altera DE-1 board.

Group 19
Frank Luong 260481340
Jeffrey Leung 260402139

References:

1. http://www.flutetunes.com/tunes/the-legend-of-zelda-theme.pdf
2. McGill University, ECSE-323, Lab 1 to 5 instructions by Prof. J. Clark.