Musical Keyboard

Roger Grayson
Tony Undercoffer
12-11-2002

Table of Contents

Chapter 1
Basic description of Musical Keyboard

We built a digital system that outputs a musical note, for ¼ of a second, when a key on the keyboard is pressed. This digital system also has volume control, balance control, and a selectable triangular or square wave output.

The keys Q through I in the first row, A through K in the second row, and Z through N in the third row each play a frequency when pressed. The frequencies are in order, with Q being the lowest frequency and N being the highest.

The volume is controlled by the + and – keys (the shift key does not need to be held down for the + key). When the + key is pressed, the volume on both channels is increased by one level, provided the volume is not all the way up. Likewise, when the – key is pressed, the volume on both channels is decreased by one level, provided the volume is not all the way down. The volume of both the left and right channels can be viewed on the LEDs on the bread board. The green LEDs are the left channel volume, and the red LEDs are the right channel volume.

The balance is controlled by the < and > keys (again, the shift key does not need to be held down). When the < key is pressed, the volume on the left channel is increased by one level, and the volume on the right channel is decreased by one level. When the > key is pressed, the volume on the right channel is increased by one level, and the volume on the left channel is decreased by one level. These changes can be viewed on the volume LEDs.

To select between the triangular and square wave output, the spacebar is pressed. When the musical keyboard is first started, the default output wave is the square wave. The lone green LED on the breadboard shows what the current output wave is: if it is lit, the output is a triangular wave. If it is not lit, the output is a square wave.

The reset button on the Xstend board is used to reset the entire circuit.

Chapter 2
Technical description of Musical Keyboard

Data Path

For the following description of the circuit, refer to the schematic on page 4.

When designing this system, the first thing we had to accomplish was to successfully receive the key from the keyboard when a key was pressed. Keyboards send three packets each time a key is pressed: a make code, a break code, and a scan code. Each of these packets is 11-bits: 1 start bit, 8 bits of data, 1 odd parity bit, and 1 stop bit. Keyboards also have their own clock, which, unlike other clocks, remains high when the keyboard is not transmitting data. This clock pulses for every bit of data that is sent, so when a key is pressed, the clock will pulse 33 times. In order to capture the scan code of the key that was pressed, we used an 11-bit shift register, clocked it with the keyboard clock, and told it to always shift right. Thus, when a key is pressed, the shift register will shift through 33 bits of data, retaining the last 11 bits, which is the scan code for the key that was pressed.

To let the rest of the circuit know that it is receiving data from the keyboard, we created a busy signal. Basically, this is just a counter and two comparators. The counter, clocked via the keyboard clock, counts up every time the keyboard sends 1 bit of data. Since the keyboard sends 33 bits, we compare the output of the counter with 32. When it is equal, the counter should reset. The output of the counter is also compared with zero. When the output is not equal to zero, the keyboard is transmitting data and the busy signal is equal to 1.

Now that we have the key that was pressed stored in the shift register, we need to determine what is done next. In this case, we have a series of comparators to determine what key was pressed. If it was one of the control keys (space, +, -, <, or >) then the corresponding comparator sends a status bit to the control unit. If it was not a control key, the scan code is processed by the keyboard to frequency decoder. This decoder converts the scan code into a 24-bit frequency. The first bit of the output of the decoder is a valid bit. If the key pressed was Q through I, A through K, or Z through N, the decoder tells the control unit that it is a valid key and send its corresponding frequency. Otherwise, the decoder tells the control unit that the key was invalid and does not output a frequency.

The frequency is then sent to two different multiplexors. The first mux has the frequency as one input and $^1/_{32}$ of the frequency as the second input. The second mux has ½ the frequency as one input and $^1/_{64}$ of the frequency as the second input. The reason this was done is the outputs of those two muxes are used in a clock divider to output a slower clock.

The clock divider is just a large counter (24-bits) and two comparators. The first comparator tells the counter to reset when it reaches a certain value, which is the output of the first mux. The second comparator compares the counter to the output of the

second mux. It outputs the slower clock, which is sent to a d flip-flop to remove any glitches.

Next we need to generate a square and triangular sound wave. Before we explain how that was accomplished, we need to explain what these waves are. A square sound wave alternates between two values: it's maximum value and it's minimum value. A triangular wave moves linearly between it's maximum and minimum values, with many values in between. The following picture illustrates both of these waves.

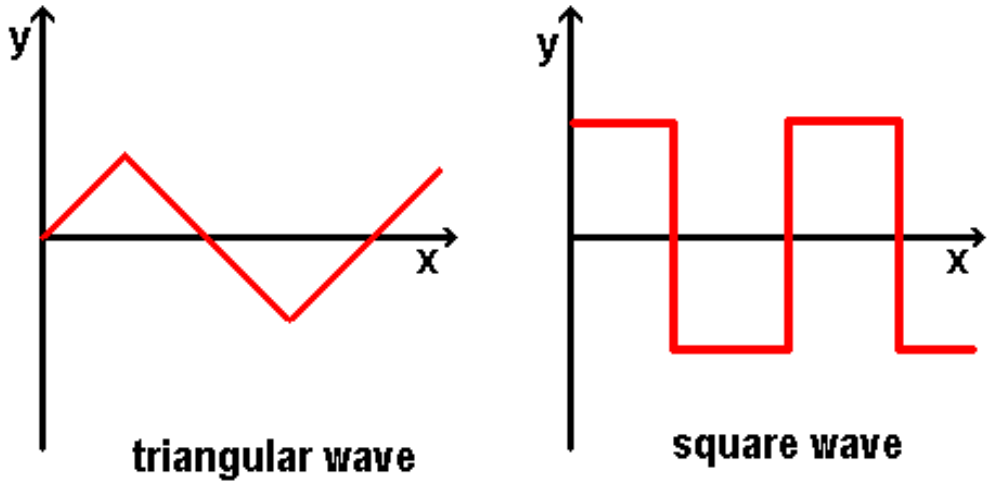

Figure 1: Triangle and Square Waves

The square wave was easy to produce since it only has two values. Since the stereo codec outputs a 20-bit two's complement number, a mux was used with a maximum value of 2^{18} and a minimum value of -2^{18}. The control for this mux is the slow clock, so the output of the mux changes on every clock edge.

The triangular wave was harder to produce. The way we accomplished it was to use a 32 by 1 mux with values ranging from 0, to the peak value (2^{18}), back down to 0, and then to the minimum value (-2^{18}). This mux is controlled by a 5 bit counter, which receives the slow clock. Every time the counter receives a clock pulse, it counts up by one, which changes the output of the mux. This produces an output with 32 discrete steps that resemble a triangular wave.

Both the square and triangular wave outputs go into two 8 by 1 muxes, one for the left channel and one for the right channel. The inputs to these muxes are the normal wave, the least significant 18 bits, 16 bits, 14 bits, 12 bits, 11 bits, and 10 bits of the wave, and 9 bits of the wave. They are used for volume control, thus they are controlled by two 3-bit saturation counters, 1 for each channel. The saturation counters count up or down when the volume is turned up or down, or when the balance is shifted to the left or right. They count up as long as they are not at their maximum value (111) and count down as long as they are not at their minimum value (000). This prevents the volume from going from it's maximum level to it's minimum level by turning it up. These saturation counters also control two 8 by 1 muxes that control the LEDs that display the volume.

The outputs of the volume muxes for both the square and triangular wave got into two muxes, one for both channels, that select which wave to output. These two muxes are controlled by 1-bit counters, which are controlled by the finite state machine. Pressing the space bar switches between the two waves.

The outputs of the wave selection muxes go into yet another mux. These muxes select whether to output the sound or not. They are controlled by the finite state machine, which, when a key is pressed, tells the mux to output the sound for ¼ of a second. The outputs of these muxes go into the stereo codec, which outputs the sound through the headphone jack on the Xstend board.

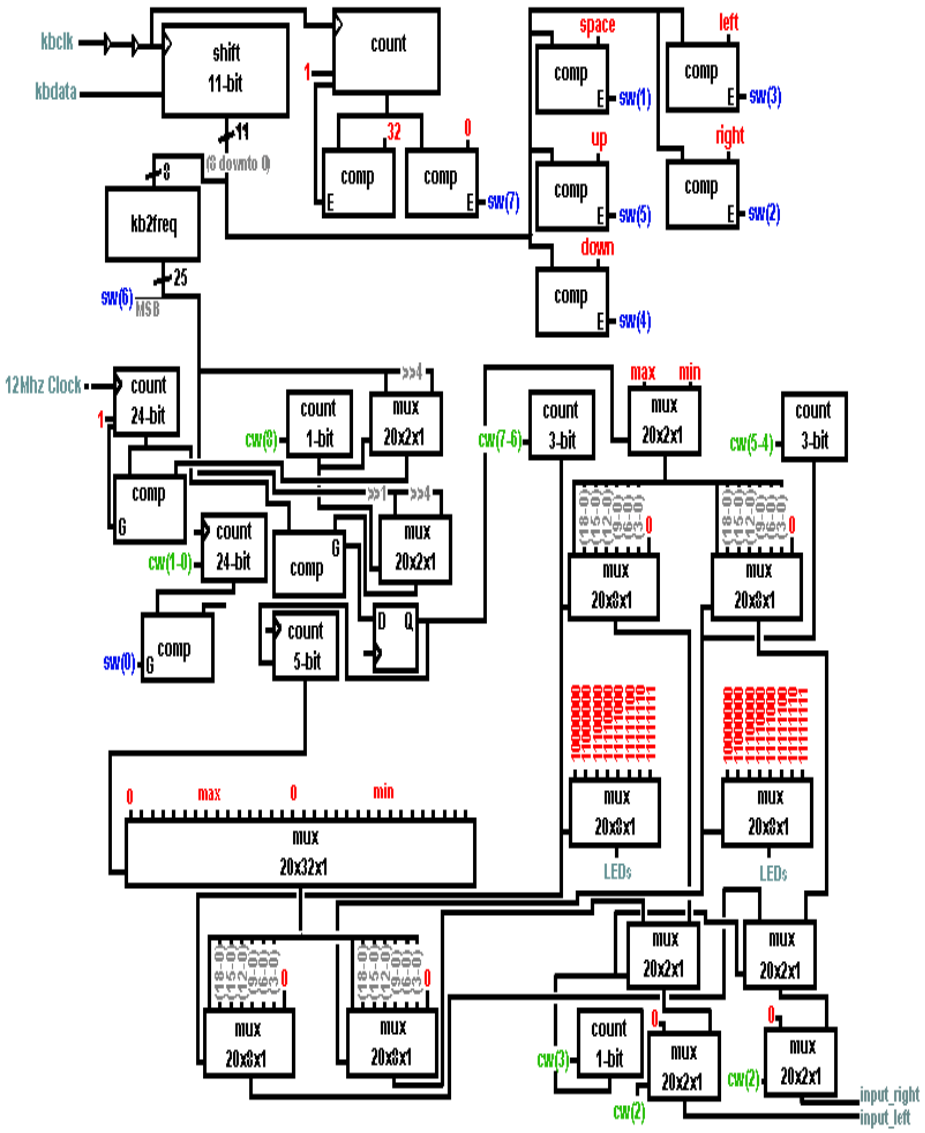

Figure 2: Data Path

Control Unit

To greatly simplify the control unit for the musical keyboard, we opted to have two control units: one for the stereo codec and another for everything else. The control unit for the stereo codec is pretty simple. Basically, the codec remains in a wait state until it is ready to output sound to the left or right channel. It then goes to the left or right state, outputs the sound, and returns to the wait state. This control unit is pictured below.

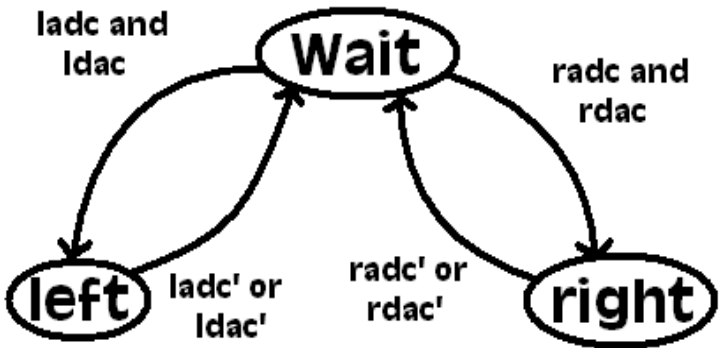

Figure 3: Codec Control State Diagram

The other control unit controls everything else: the keyboard input, the square and triangular wave selection, volume control, frequency control, and LED control. Initially, this control unit starts in the RST state, which resets all the components. It then remains in a wait state (*wait1*) until a key is pressed. Once a key is pressed, it again remains in a wait state (*wait2*) until the key is released. This is a two-line handshake with the keyboard. Once the key is released, the control unit determines what key was pressed. If the user pressed a key that is not used, the keyboard to frequency decoder will tell the control unit that it was not a valid key and it will return to the *wait1* state. If the user pressed the spacebar, the control unit goes to the *change output* state, which changes the output between the square and triangular waves by telling the 1-bit counter which controls the wave selection mux to count up. If the user pressed the > key, the control unit goes to the *inc right dec left* state, which increases the volume in the right channel and decreases the volume in the left channel by telling the saturation counter that controls the right channel to count up and telling the saturation counter that controls the left counter to count down. If the user pressed the < key, the control unit goes to the *inc left dec right* state, which increases the volume in the left channel and decreases the volume in the right channel by telling the corresponding saturation counters to count up and down. If the user pressed the + key, the control unit goes to the *inc volume* state, which increases the volume in both channels by telling both saturation counters to count up. If the user pressed the – key, the control unit goes to the *dec volume* state, which decreases the volume in both channels by telling both saturation counters to count down. If the user pressed one of the designated frequency keys, the control unit goes to the *reset ¼ S count* state, which resets the ¼ second counter and goes to the *output* state. While the ¼ second counter has not reach ¼ second yet, the output state continuously outputs the sound. Once ¼ second is reached, it stops outputting sound and returns to the *wait1* state. Below is a picture of this control unit.

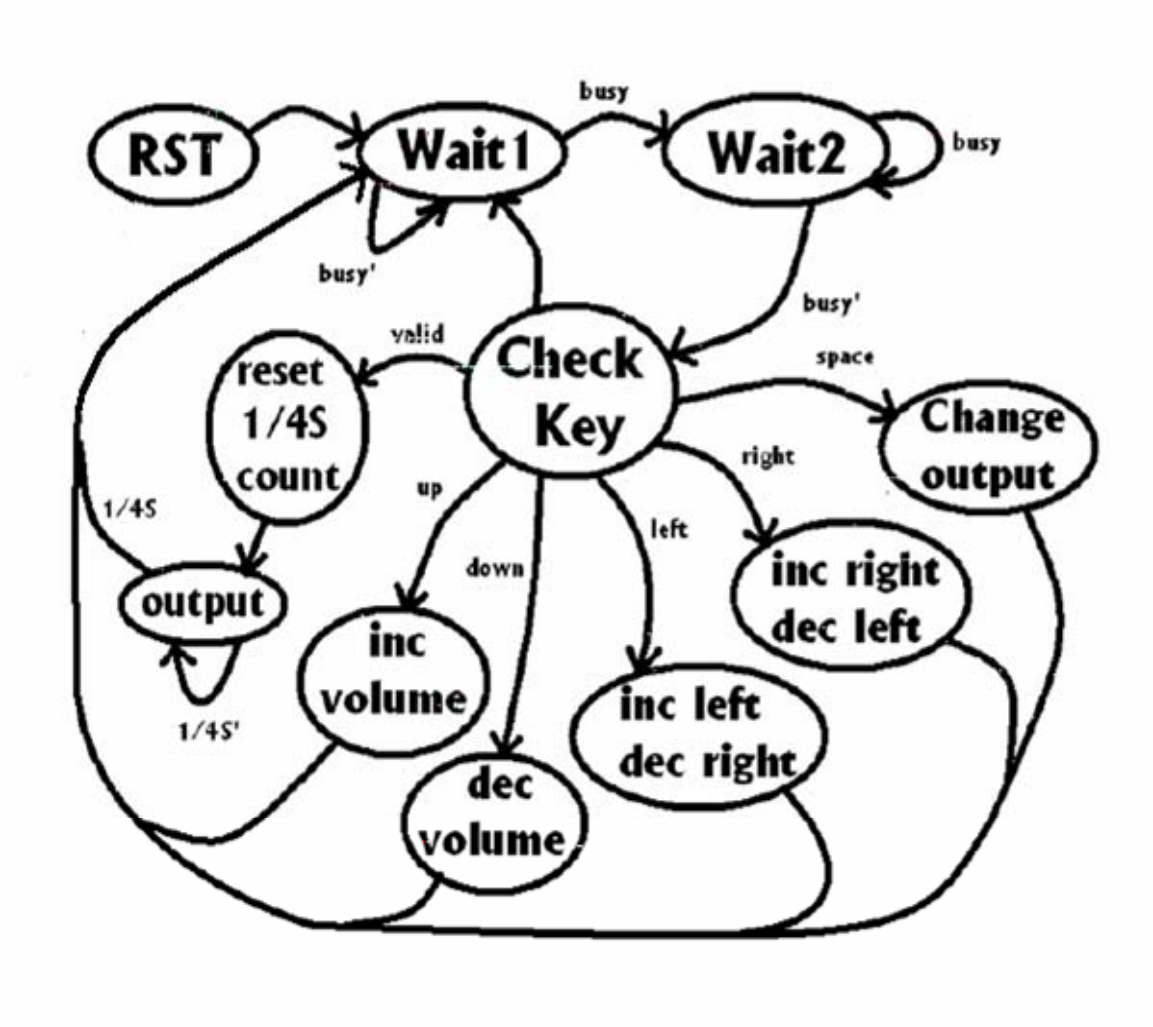

Figure 4: Musical Keyboard FSM State Diagram

Chapter 3
Unique Attributes of Musical Keyboard

When assigning the different frequencies to keys on the keyboard, we used three different octaves. In each case, the octave contained only the major scale. A major scale consists of eight different notes with the frequency of the lowest note in the octave being half of the highest frequency. Each set of consecutive notes in the major scale consists of either a one half step or two half step (full step) difference in frequency. A step is based on the ratio of the higher frequency to the lower frequency. This ratio for all half steps is 1.059. As a result, each half step up is 1.059 times larger than the frequency a half step below it. We chose the first note in the middle octave to be the equivalent of the A above the first C on a piano which has a frequency of 440 Hz The frequencies for all of keys in our circuit are listed below with values of Hz.

full step	full step	half step	full step	full step	full step	half step

Q	W	E	R	T	Y	U	I
220	250	277	294	330	370	415	440

A	S	D	F	G	H	J	K
440	494	555	588	660	740	831	880

Z	X	C	V	B	N
880	988	1109	1175	1319	1480

Another unique attribute of our circuit was using LED's to display the volume and balance of the outputs. Without adding any difficulty to the circuit, it greatly increased the appearance and visual appeal of the circuit. Rather than just giving the audio output, it gives a visual representation of what the circuit is doing allowing people to see as well as hear what is happening. The only added requirement of displaying the volume in this manner was to include a bread board with an active low LED setup and wire each to an unused pin on the XStend board. The outputs on the pins were simply determined by the value of the volume counter which controlled a mux that selected different light patterns to show the volume (This is further explained in the previous section).

Lastly, the ability to change between a square or triangle wave output added uniqueness to our project. Using both square and triangle outputs demonstrated how different waves can be used to produce relatively similar outputs. The primary factor in producing similar sounds from different waves is assuring that the time between the maximum and minimum output levels of the wave are the same for each type of sound wave. As a direct result, the amount of time between each discrete step in a triangle wave can be determined. Because we had 32 (2^5) different steps in each period, we were able to use the values for our square wave frequencies and simply shift them left by

5 bits and still maintain the same amount of time between the minimum and maximum values.

Chapter 4
Afterthoughts

While implementing the design of our musical keyboard, we encountered a few minor problems. Also, there were aspects of our project that could have been improved. The first problem we encountered was with our busy signal. Occasionally, it would get stuck at a logic one and lock our finite state machine in one of the wait states. We were not able to determine what the exact cause of the problem was, but we knew that it had to deal with the counter that generated our busy signal. For some reason, this counter was either resetting too soon or not resetting at all sometimes. This may have been due to glitches in the clock signal produced by the keyboard. One possibility of remedying this problem is to debounce this clock signal. For our purposes, it was easiest to add an extra button which could be used to reset the counter when busy got stuck. This reset was wired to the spare button on the XStend board. In order to reset the counter, the spare button had to be held down while a key on the keyboard was pressed. It is necessary to press a key on the keyboard because this specific reset was synchronous and required a clock edge and a clock edge is only produced by the keyboard when a key is pressed.

Another rather impractical aspect of our design was using a 20x32x1 mux to generate the triangle wave. This is obviously not the most efficient way to implement this type of wave although it may be the easiest. It requires a very large amount of gates and much of the Xilinx chip's resources. Another way to implement this aspect of the design would have been to use an adder/subtracter and two comparators with the output of the adder/subtracter latched up by a register and fed back into the adder. Each time the clock edge hit, the register would load a new value (the output of the adder/subtracter). The value of the register would be continually increased by a discrete step size until it reached its maximum value. At this time, a comparator would tell the adder/subtractor to subtract until the minimum value was reach. At this point, another comparator would change the function back to addition. This could be done by ORing the equal outputs of the two comparators together and having them control a 1-bit counter with its output controlling the function bit of the adder/subtractor. The output of the register would be sent to the codec interface as the triangle wave signal.

We encountered another problem when trying to convert from scan codes to frequencies. For some reason, Xilinx would only allow a maximum of 22 conversions using our design for the scan code to frequency converter. We wanted 24 different frequencies which we needed to obtain all 3 octaves. Unfortunately, we were unable to determine the cause of this problem and simply left the highest octave incomplete with 6 different notes instead of 8.

Another change we could have made is to add minors to our octaves. Because we were limited to the major scale, there were very few songs that could be played on the keyboard. Adding more minors, which involves adding more half steps (explained in chapter 3) would have completed each octave. However, each octave would have required 13 different frequencies instead of just 8 which our design used. This may limit

the number of full octaves because we were only able to produce 22 frequencies as I explained previously.

The last improvement to our design could have been to add an exponential wave to give an example of one more sound wave and determine any differences in sound that it may have produced

Appendix A
Duplicating the Demo

Included on the CD attached at the end of this document is a working bit file for our design. Along with this, there are a few other things that need set up to create a complete working demo. We found that using the PS/2 port on the XS40 board worked better than the PS/2 port on the XStend board. For some reason, it created fewer problems with our busy signal. However, if the busy signal does still get stuck, the procedure explained in chapter 4 can be used to reset it (the busy signal is displayed on the bottom bar of the left seven-segment display. When the bar remains on, the busy signal must be reset).

For this circuit, only the output channel of the stereo codec was needed. Obviously, speakers must be hooked to this port to generate the sounds. However, we found that speakers without any amplification could not produce very loud sounds.

The last step is to set up the LED's for the volume display. Each LED is wired in an active low configuration using the 5v source on the XStend board and 1.8k resistors. The setup for each LED is shown below.

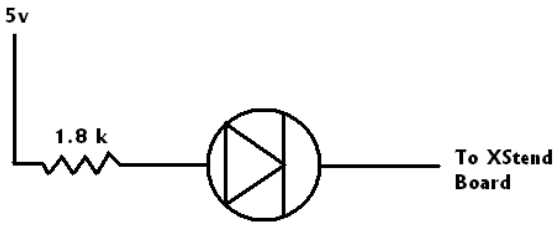

Figure 5: Active Low LED Configuration

Each LED is wired to a different pin on the XStend board. However, the numbers beside each pin are not the correct numbers. The actual numbers are on the XS40 board. The corresponding pin is directly across from these numbers. The pins for the LED's for the left volume, right volume, and square or triangle status are included below.

L1: pin 10	R1: pin 84	SquareOrTriangle: pin 57
L2: pin 80	R2: pin 83	
L3: pin 81	R3: pin 82	
L4: pin 35	R4: pin 79	
L5: pin 38	R5: pin 78	
L6: pin 39	R6: pin 5	
L7: pin 40	R7: pin 4	
L8: pin 41	R8: pin 3	