The LightBox

Final Project Report
December 8, 2006
E155 – Microprocessor-Based Systems

Dane Lindblad and Neel Shah

Abstract:

Synchronized lighting is essential to an audiovisual experience or to amplify a party atmosphere. Traditionally, lighting is done manually or a customized sequence is manually generated for each song. The objective of this project is to produce a LightBox that takes in any audio input and outputs a synchronized light display. The project consists of an analog filtering and rectifying unit, an A/D conversion unit running on a PIC, and a keypad controller and light sequence bank synthesized on an FPGA. The user hooks up his or her audio output to the LightBox, and the LightBox outputs a unique light pattern based on the audio and the user's numerical input from 00-99 on the keypad.

Introduction

There was an item of particular interest at a private Pomona party earlier this year. The host claimed that he bought it in a Chinese black market for five-hundred dollars. Neel claimed that he could build one for one-tenth that price. And like that, our final project was chosen. The item in question had a power input, left and right audio inputs, multiple power outputs, and a knob. Whatever was inside the box analyzed the audio signal inputs, then, based on the position of the knob, would output power to a selection of the power outputs. Into these outputs the host had plugged lights of varying forms and colors. The result was a custom light show, based on the music, and the LightBox would be an object that would be fantastic for any dorm room.

The objective of this project is to produce a LightBox that takes in any audio input and automatically outputs a synchronized light display. Unlike manually generated light displays, the LightBox is automatic; unlike the item at the Pomona party, the LightBox prototype costs $0 to develop. The LightBox consists of an analog filtering and rectifying unit, an A/D conversion unit running on a PIC, and a keypad controller and light sequence bank synthesized on an FPGA. The user hooks up his or her audio output to the LightBox, and the LightBox outputs a unique light pattern based on the audio and the user's numerical input from 00-99 on the keypad.

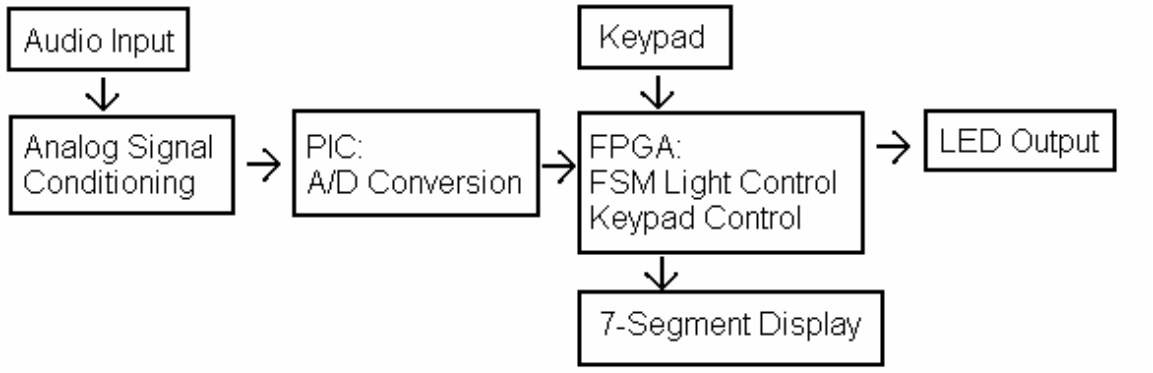

Figure 1: Block Diagram Overview of the LightBox.

Figure 1 displays a block diagram overview of the LightBox. There are two different signal paths for two different user inputs. The audio input gets filtered (much like an equalizer) and rectified through analog circuitry, converted to a digital signal with the PIC, and used as a digital input to the FSM light control on the FPGA. The FSM light control outputs another digital sequence used to power lights (in this case, LEDs). The keypad input is sent to the FPGA and the result is sent to a dual seven-segment display. This is the user interface for the LightBox that shows which FSM light sequences has been selected.

Breadboard Schematics: Analog Signal Conditioning

The analog portion of the LightBox prototype is placed on a utility breadboard. The analog signal conditioning can be broken into three separate parts. The first part is a summing amplifier that combines the left and right channels into one input signal and amplifies it to an appropriate level. The second part is a set of eight active band pass filters that filter the audio signal over selected frequency ranges. The third part is a set of eight rectifiers that convert the audio signal to DC voltages that spike when the volume over a certain frequency range increases. The output from this third part is what gets sent to the eight A/D conversion channels on the PIC. Figure 2 displays a block diagram of this subunit. The analog signal conditioning subunit runs off a ±20 V DC power supply. All op-amps in the subunit are 411 op-amps running off of $V_+ = 20$ V and $V_- = -20$ V.

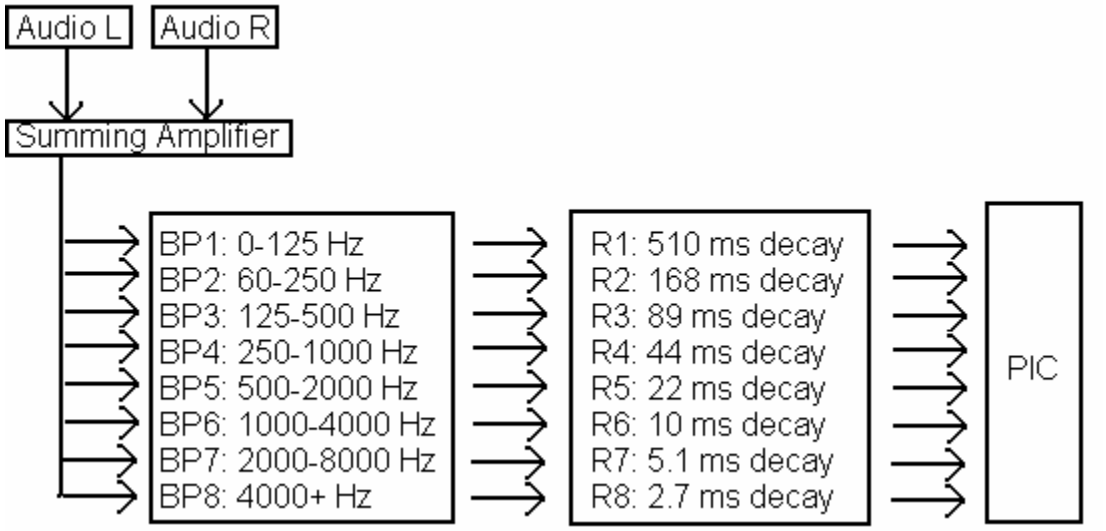

Figure 2: Block Diagram overview of the analog signal conditioning unit located on the breadboard. BP denotes a band pass filter; R denotes a rectifier.

A more detailed schematic of the summing amplifier is shown on the next page in Figure 3. The summing amplifier consists of one 411 op-amp and three resistors. The channels are combined due to the constraints on the PIC A/D conversion, which only accepts 8 analog signals. Rather than divide the left and right channels into only four frequency bands, the channels were combined in order to obtain filter eight frequency bands.

The resistor R_0 was adjusted to a value of 5.6 kΩ in order to boost the max input signal value from 0.75 V to 4.2 V (standard inverting amplifier math). Note that while the sources in the diagram below are in phase, the audio channels are rarely in phase and the two signal sources were rarely found to be additive and in phase. Thus the max input signal value is taken to be 0.75 V instead of 1.5 V. This was verified by looking at an iPod stereo input signal via oscilloscope.

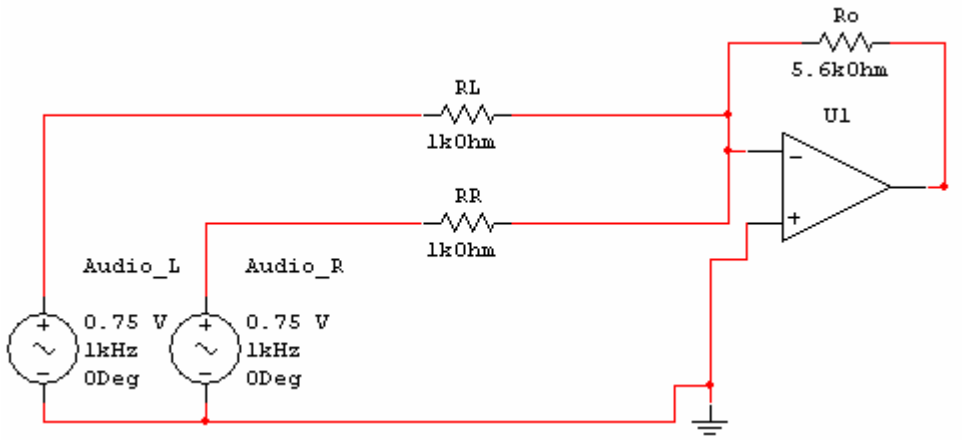

Figure 3: Schematic of summing amplifier combining and amplifying left and right audio signals.

The second element of the analog subunit is the band pass filter array. All eight band pass filters are in parallel and process different bandwidths. Figure 4 displays one sample band pass filter for the 125 Hz – 500 Hz frequency band. Table 1 at the end of this section contains all the values for the band pass filters.

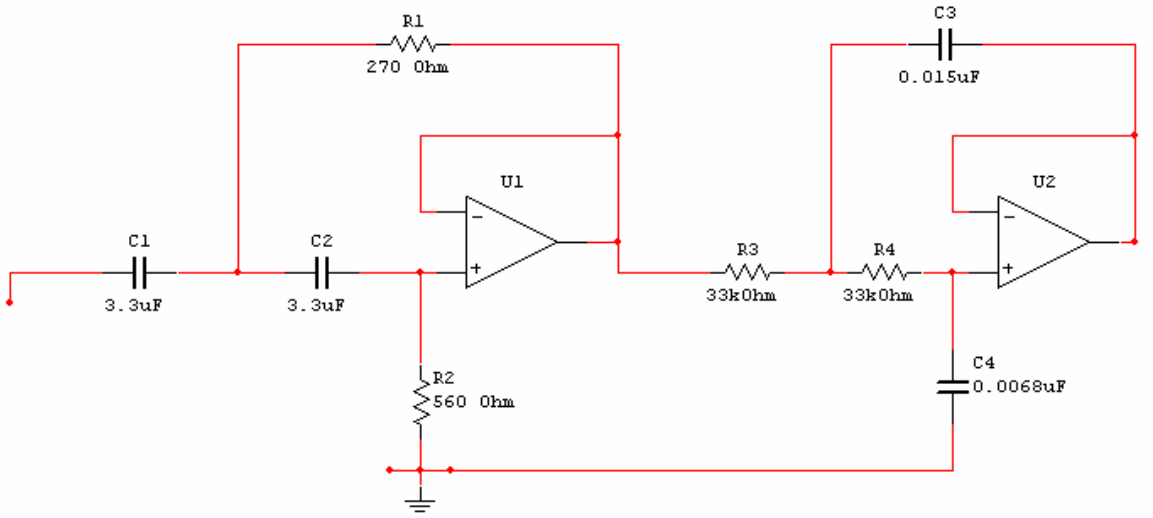

Figure 4: Band pass filter for the 125 Hz to 500 Hz frequency band. The first op-amp unit is a Butterworth high-pass filter with a -3 dB point calibrated to ~125 Hz, while the second op-amp unit is a Butterworth low pass filter calibrated to 500 Hz.

The magnitude of the voltage transfer function of the band pass filter is

$$|T| = \frac{1}{\sqrt{1+\left(\frac{f}{f_{3dB,LP}}\right)^4}} \times \frac{1}{\sqrt{1+\left(\frac{f_{3dB,HP}}{f}\right)^4}} \qquad (1)$$

For frequencies between the low pass 3 dB point ($f_{3dB,LP}$) and the high pass 3 dB point ($f_{3dB,HP}$), |T| is approximately 1. Outside this range, |T| is effectively 0.

The third element is the rectifier unit. Figure 5 shows the corresponding rectifier for the band pass filter shown in Figure 4. A diode drops the voltage level from 4.2 V to 3.5 V and removes the negative half-cycle of the sine wave. An RC unit performs the rectification and the voltage is capped at 3.3 V by a Zener diode. The RC value was chosen such that the RC time constant (which translates to decay time) is about ten times the lowest expected frequency. These values are also included in Table 1.

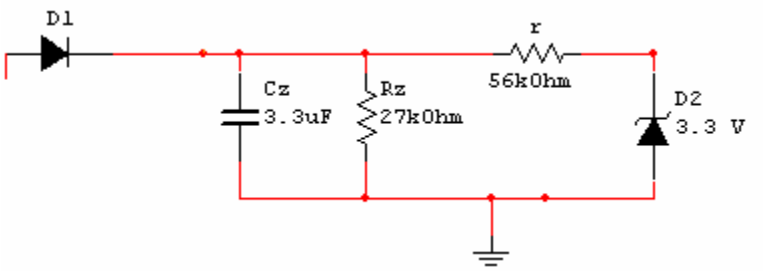

Figure 5: Rectification (DC conversion) subunit for band pass filter in Figure 4.

The 56 kΩ resistor regulates the power consumption by the Zener diode so it will not break down. (Standard Zener power limits are approximately 250 mW). The output is taken over the Zener diode and input into the appropriate PIC A/D channel.

Table 1 below shows all the component values for the second and third stages for the analog signal conditioning subunit. R_Z and C_Z values are tuned to produce an RC value approximately 10 times greater than the period of the lowest frequency in the band. Other component values chosen by picking RC values that correspond to selected 3 dB points that also yielded values of R_1/R_2 and C_1/C_2 that were sufficiently close to standard values.

	Highpass freq (Hz)	Lowpass freq (Hz)	R (kΩ)	R1 (kΩ)	R2 (kΩ)	C (μF)	C1 (μF)	C2 (μF)	Rz (kΩ)	Cz (μF)	RC (ms)
Filter 1		125	0.39				4.7	2.2	51	10	510
Filter 2	60	250	62	18	8.2	0.22	0.015	0.0068	51	3.3	168
Filter 3	125	500	33	0.56	0.27	3.3	0.015	0.0068	27	3.3	89.1
Filter 4	250	1000	47	91	47	0.01	0.0047	0.0022	20	2.2	44
Filter 5	500	2000	24	47	24	0.01	0.0047	0.0022	10	2.2	22
Filter 6	1000	4000	12	68	33	0.0033	0.0047	0.0022	10	1	10
Filter 7	2000	8000	6.2	33	18	0.0033	0.0047	0.0022	5.1	1	5.1
Filter 8	4000			18	8.2	0.0033			2.7	1	2.7

Table 1: Component values for parallel band pass filters and rectification units.

Microcontroller Design: PIC A/D Conversion

The overall concept for the PIC is straightforward. The code for the PIC is in Appendix C. The program consists of three stages. The first stage consists of sampling all 8 analog channels, and writing them to individual locations in a table. Since the signals will be rectified before entering the PIC, the delay induced by sampling will be negligible in terms of the signal. This stage also allows for digital manipulation of voltage levels in order to weigh some channels more heavily than others. If a channel is experiencing high volume noise, its multiplier can be lowered so it does not affect the threshold calculations and subsequently does not light. Once all eight channels have been sampled and multiplied, in stage 2 the threshold voltage is determined by averaging all the input values. Stage 3 consists of writing the Boolean values for each channel to the outputs of the PIC on PORTC to the FPGA where they are accepted as the bus ChannelIn.

The eight analog channels are located on PORTA (RA0,1,2,3,5) and PORTE (RE0,1,2). RA0 corresponds to AN0, and RE2 corresponds to AN7. AN0 corresponds to the lowest frequency band, while AN7 corresponds to the highest frequency band. TRISA and TRISE are thus set to 0xFF. Since the output is being sent to PORTC, TRISC is set to 0x00. ADCON1 is set to 0x81 while ADCON0 starts at 0x80. This right-justifies the A/D conversion and sets the appropriate clock ($F_{osc}/32$ for a 20 MHz Harrisboard clock).

FPGA Design

The third section of the LightBox is the FPGA, of which there are two main divisions. The User Input section is a keypad interface similar to that from Lab 4. The FSMs section is what actually coordinates the lightshow.

User Input

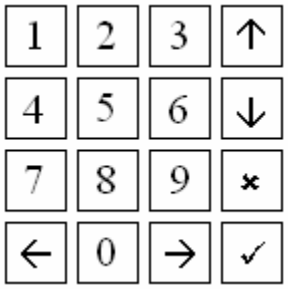

Figure 6 - Adapted Keypad

All the workings of the keypad are the same as they were for Lab 4. There is a FSM polling all the columns, forcing one of them at a time to 0, and reading the rows. If any of the rows are 0 the FSM pauses and the keypad recognizes that a button has been pressed. The current rows and columns are fed into another module (`crtonumber`) which interprets whether the current combination of values for rows and columns is a valid keystroke, and what key it should be interpreted as. This information is fed into another module (`interpretnumber`) which is the module that determines what the hardware should do when that button is pressed. The entry system has a few different states. These are described below:

State 0

State 0 is the state that the display is in when the last button pressed was ✓ or × (confirm or cancel). While in this state the readout displays the number of the active FSM. Pressing any button besides ✓ or × will move the entry system to another state. If the button pressed is 0-9 or ←, the system moves to State 1. If ↑, ↓, or → is pressed, the system goes to State 2.

State 1

State 1 is the state in which the user enters the tens digit. If 0-9 is pressed, it changes the displayed digit to that number. If ↑ or ↓ is pressed the displayed number is incremented or decremented (this feature tends to be buggy). While in this state, the left digit of the readout flashes on and off to indicate it is being edited. Only the →, ✓, or × buttons will move the system out of State 1. If the ✓ button is pressed the current displayed values become the new active values, and the system returns to State 0. If the × button is pressed, the displayed digits are replaced with the old active values and the system returns to State 0. If the → button is pressed, the system goes to State 2.

State 2

State 2 is the state in which the user enters the ones digit. It is essentially the same as State 1, except the → button no longer does anything, and the ← moves the system back to State 1.

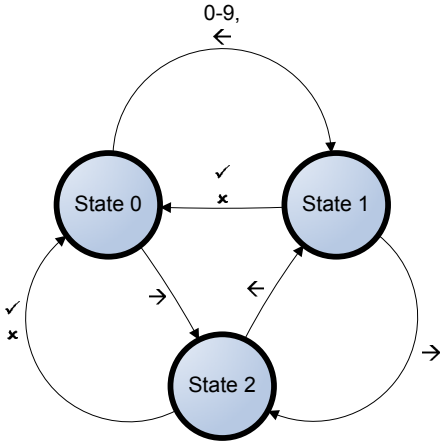

Figure 7 - Keypad State Diagram

Interpretation Hardware

There are a few features that are necessary to make the hardware function that way. There is a register to designate whether the system is in State 0 or not (`inprogress`) and another to specify, if it is not in State 0, which state it is in (`digselect`). There is a third register to hold the value of the digit that is currently being edited (`diginput`). There are also registers for the cancel and confirm signals. There are also flip-flops for the output values being used (`onesdigit`, `tensdigit`), and the values being displayed (`onestemp`, `tenstemp`). The way these are used is demonstrated in the figure below.

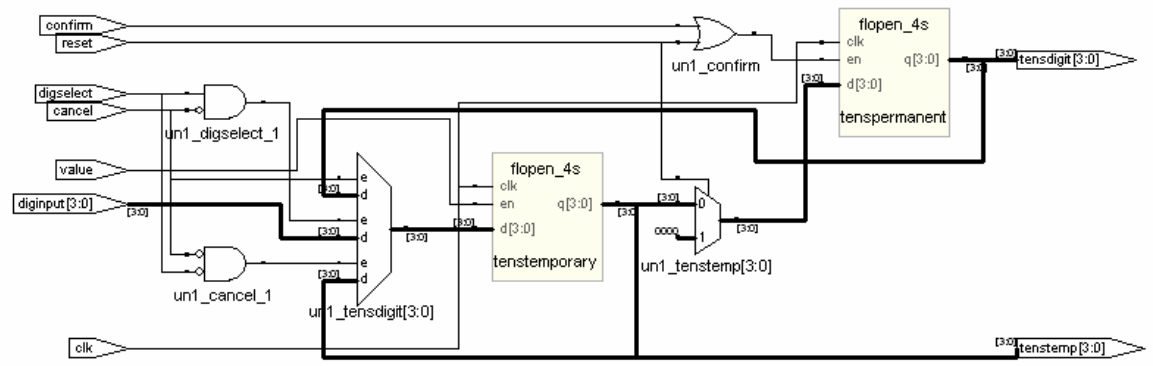

Figure 8 - Tens Digit Registers

Blinking Digit

With the outputs for the tens digit being used, and the tens digit being displayed, and the registers for the digit, and whether it is being edited, the only problem left is how to make the digit blink. The method used is shown in the following figure.

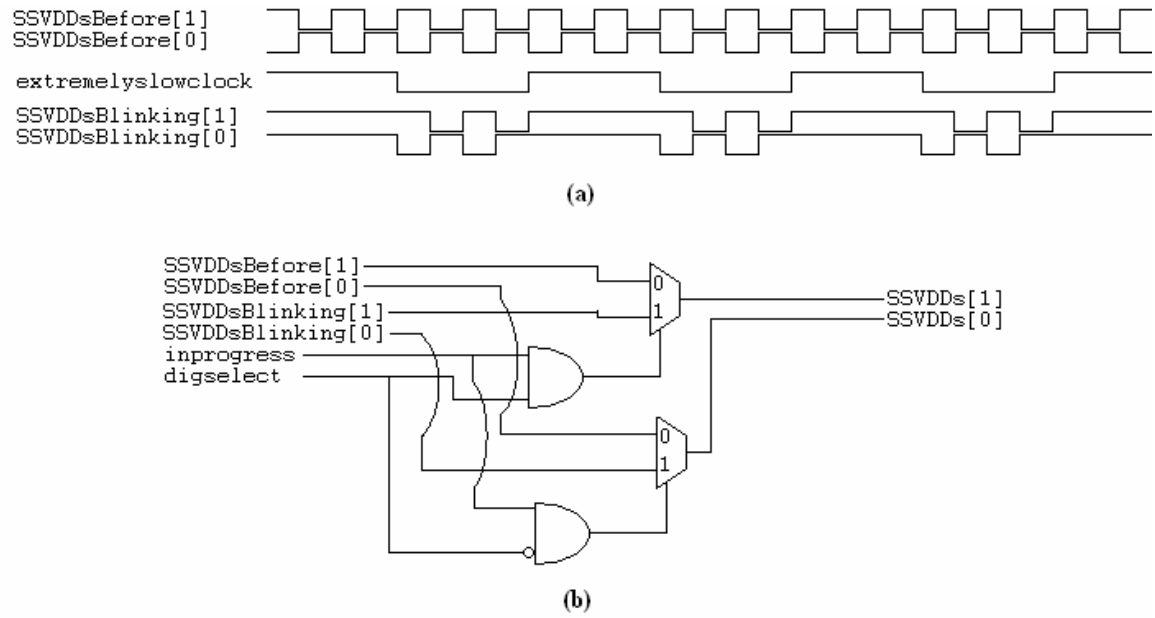

Figure 9 - Blinking SSVDDs

In part (a), `SSVDDsBefore` is the `SSVDDs` signal generated by Lab 3 for multiplexing displays. It is the [8] bit of a clock counter in the `controlunit` module. Since the displays used are common anode displays, to turn them off, the SSVDD signal needs to go high for that digit when the light should be off. This period was shortened in this graphic, but in the code it corresponds to the [26] bit of a clock counter. The `SSVDDsBlinking` signal is the OR of `SSVDDsBefore` and `extremelyslowclock.`

Part (b) of Figure 9 shows how the `SSVDDs` signal for each channel digit is selected. If `inprogress` is high and `digselect` indicates the digit in question, its `SSVDDs` signal is blinking.

FSMs

Even though the FSMs module takes up over 85% of the Verilog code (20 out of 26 pages), it is relatively simple to explain. There are really only two parts to the module. The first part takes the user input number, and determines what the next state will be, and which register the lights should be taking their output from. The second part takes the current audio triggers and determines which registers should take the current next state value.

Next State Determination

This section of the code basically contains a case statement based off the user input. For each case there is a different FSM, with a case statement based on the current state of the lights. The majority of the FSMs assign a specific light configuration to the `nextstate` register and assigns that the lights should read from a specific register, although some just connect the output directly to the input. The purpose of assigning the lights to a register will become clear in the next section.

State Change Cues

This section of the code consists of a few different always blocks. Each block takes a different type of cue for a different type of FSM advancement. This is needed so the FSMs can take cues from different channels. One side effect of this is that, since it is only to assign values to a register within one always statement, one register is required for each always block. This is why the statement determining which register to read from is necessary. Some of the blocks just assign the `nextstate` to the effective state register, while others contain case statements for special treatment of individual FSMs.

Results

The analog circuitry was designed fairly rapidly and easily. The most difficult part of this subunit was arranging all the components in a logical and space efficient manner on the breadboard, which was accomplished after careful planning. This made troubleshooting for the occasional shorted op-amp extremely easy. The summing amplifier worked as expected, raising the input signal from 0.75 V_P to 4.2 V_P. The filters worked as expected. The oscilloscope FFT function was used to verify each filter was functional within its frequency range. A major anomaly that was resolved was the introduction of high voltage, high frequency noises by the op-amps. When viewing the outputs of an op-amp through an oscilloscope, occasional bursts of noise outside the audio band were visible. This noise would extend to -20 V, the value of V_- for the op-amps. It was concluded the ±20 V DC source was the source of the anomaly as well and the issue was resolved by decoupling the power sources with 2.2 µF electrolytic capacitors.

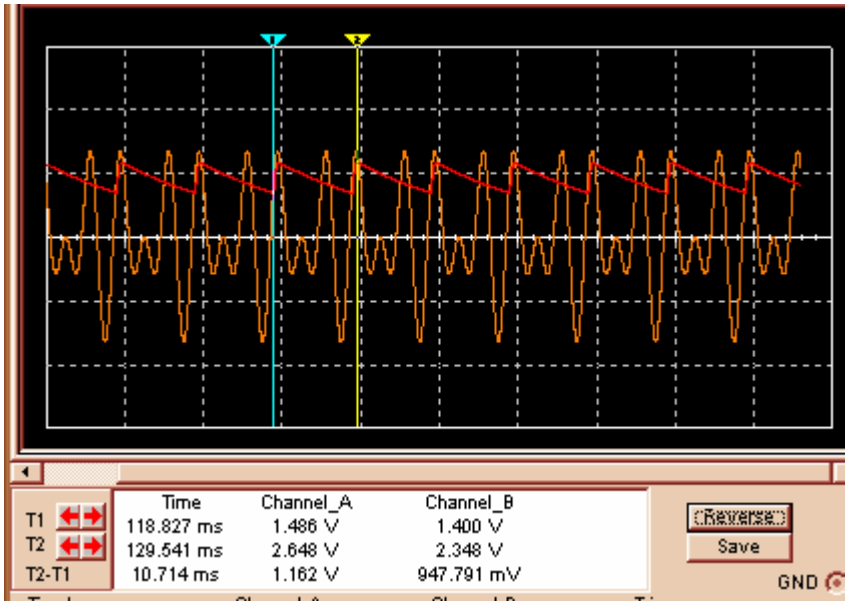

Figure 10: Multisim simulation of the 125 Hz - 500 Hz analog signal conditioning subunit. Amplified input (orange) peaks correspond to PIC input (red) peaks that die off until the next peak.

An unresolved analog issue was the voltage regulation of the Zener diode. The Zener diodes were procured from the Engineering stockroom and were clearly marked as 3.3 V Zener diodes. However, the diodes capped the voltage around 1.5-2.0 V. This was observed by applying the oscilloscope on both ends of the **r** = 56 kΩ (refer to Figure 5). To the left of **r**, the voltage levels

were reading as expected, around 3.5 V. To the right of **r,** voltage levels did not exceed 2.0 V, and rarely strayed above 1.7 V. Different values of **r** were experimented with, including 33 kΩ and 24 kΩ, with no change. The issue was left unresolved since all eight Zener diodes displayed this characteristic. Since all channels were being scaled down evenly, the PIC algorithm would still operate the same and no one channel would be compensated (although sensitivity and range is reduced).

Finally, the frequency bands being filtered need to be tinkered with in order to better capture instruments. The frequency bands being currently used are generic; based simply on an equalizer. The lowest two channels rarely capture anything musical. The components of these filters can be changed to focus more sharply around a certain frequency. Although instruments with wide ranges of pitches and spectral envelopes (such as an electric guitar) are difficult to consistently capture, percussive instruments and bass instruments are far easier to capture (as evidenced by the successful playback of "Seven Nation Army" in FSM 30). Sometimes bass instruments interfere with bass drums on the same channel and there is no small fix to separate them.

The PIC was programmed quickly and easily and worked correctly on the first test. The same A/D conversion code known to work was lifted from Lab 7 and inserted in a `for` loop to read eight channels sequentially. The threshold calculation and output values also performed as expected. The PIC was tested by using 7 1 kΩ resistors in series to create eight voltage divisions. Table 2 details the theoretical voltage values, measured voltage values, and digitally recorded PIC values for the test case. The values are within 60 mV of each other, which is permissible for a fairly low-sensitivity application.

Theoretical Voltage (V)	Measured Voltage (V)	PIC Voltage (V)	Error (mV)
3.300	3.3	3.300	0
2.829	2.8	2.792	-36
2.357	2.4	2.375	18
1.886	1.9	1.906	20
1.414	1.4	1.392	-22
0.943	0.9	0.887	-56
0.471	0.5	0.492	21
0.000	0.0	0.010	10

Table 2: Data for PIC test. Theoretical Voltage indicates expected value according to voltage division, Measured Voltage indicates analog voltage when measured with oscilloscope, PIC Voltage indicates PIC A/D results.

FPGA

Appendix A: FPGA Verilog Code

```
module Top(clk, reset, ChannelIn, columns, rows, SSVDDs, SSout, LightsOut);
    input clk;
    input reset;
    input [7:0] ChannelIn;
    output [3:0] columns;
    input [3:0] rows;
    output [1:0] SSVDDs;
    output [6:0] SSout;
    output [7:0] LightsOut;

    wire [7:0] PNUM;
    wire [5:0] columnsbefore;
    wire [4:0] sum;
    assign columns = columnsbefore[3:0];
Main fsms(ChannelIn, LightsOut, clk, PNUM, reset);
lab4 userinput(rows, columnsbefore, clk, reset, SSout, SSVDDs, sum, PNUM);
endmodule

module Main(ChannelIn, LightsOut, clk, PNum, reset);
        input [7:0] ChannelIn;
        output [7:0] LightsOut;
        input clk, reset;
        input [7:0] PNum;

reg [9:0] state00, state01, state02, state03, state04, state05, state06, state07;
reg [9:0] nextstate;
reg [9:0] state;
assign LightsOut = state[7:0];
wire [31:0] rand;
reg repeatoff;
reg [2:0] sum;
counter #(32) randnumber(clk, 1'b1, 1'b0, 32'b0, rand);
assign slowerclock = rand[24];
reg [2:0] randn1, randn2, randn3;

always@( posedge clk )
if(reset) state <= 10'b00_0000_0000;
else
case (PNum)
        8'b0000_0000: state[7:0] <= ChannelIn;
        8'b0000_0001:
                begin
                        case (state[7:0])
                                8'b00011000:   nextstate[7:0]<= 8'b00100100;
                                8'b00100100:   nextstate[7:0]<= 8'b01000010;
                                8'b01000010:   nextstate[7:0]<= 8'b10000001;
                                8'b10000001:   nextstate[7:0]<= 8'b00011000;
                                default:       nextstate[7:0]<= 8'b00011000;
                        endcase
                        state          <= state01;
                end
        8'b0000_0010:
                begin
                        case (state [7:0])
                                8'b00011000:   nextstate[7:0]<= 8'b10000001;
                                8'b00100100:   nextstate[7:0]<= 8'b00011000;
                                8'b01000010:   nextstate[7:0]<= 8'b00100100;
                                8'b10000001:   nextstate[7:0]<= 8'b01000010;
                                default:       nextstate[7:0]<= 8'b10000001;
                        endcase
                        state <= state01;
                end
        8'b0000_0011:
                begin
                        case (state[8:0])
                                9'b010000001:  nextstate <= 10'b0001000010;
                                9'b001000010:  nextstate <= 10'b0000100100;
                                9'b000100100:  nextstate <= 10'b0000011000;
                                9'b000011000:  nextstate <= 10'b0100100100;
                                9'b100100100:  nextstate <= 10'b0101000010;
                                9'b101000010:  nextstate <= 10'b0010000001;
                                default:       nextstate <= 10'b0010000001;
                        endcase
                        state <= state01;
                end
```

```
8'b0000_0100:
        begin
                case (state[8:0])
                        9'b010000001:  nextstate <= 10'b0001000010;
                        9'b001000010:  nextstate <= 10'b0000100100;
                        9'b000100100:  nextstate <= 10'b0000011000;
                        9'b000011000:  nextstate <= 10'b0100011000;
                        9'b100011000:  nextstate <= 10'b0100100100;
                        9'b100100100:  nextstate <= 10'b0101000010;
                        9'b101000010:  nextstate <= 10'b0110000001;
                        9'b110000001:  nextstate <= 10'b0010000001;
                        default:       nextstate <= 10'b0010000001;
                endcase
                state <= state01;
        end
8'b0000_0101:
        begin
                case (state[8:0])
                        9'b000011000:  nextstate <= 10'b0000111100;
                        9'b000111100:  nextstate <= 10'b0001111110;
                        9'b001111110:  nextstate <= 10'b0011111111;
                        9'b011111111:  nextstate <= 10'b0111111111;
                        9'b111111111:  nextstate <= 10'b0101111110;
                        9'b101111110:  nextstate <= 10'b0100111100;
                        9'b100111100:  nextstate <= 10'b0000011000;
                        default        nextstate <= 10'b0000011000;
                endcase
                state <= state03;
        end
8'b0000_0110:
        begin
                case (state[8:0])
                        9'b000000000:  nextstate <= 10'b0000011000;
                        9'b000011000:  nextstate <= 10'b0000111100;
                        9'b000111100:  nextstate <= 10'b0001111110;
                        9'b001111110:  nextstate <= 10'b0011111111;
                        9'b011111111:  nextstate <= 10'b0111111111;
                        9'b111111111:  nextstate <= 10'b0101111110;
                        9'b101111110:  nextstate <= 10'b0100111100;
                        9'b100111100:  nextstate <= 10'b0100011000;
                        9'b100011000:  nextstate <= 10'b0000000000;
                        default        nextstate <= 10'b0000000000;
                endcase
                state <= state03;
        end
8'b0000_0111:
        begin
                case (state[8:0])
                        9'b010000000:  nextstate <= 10'b0011000000;
                        9'b011000000:  nextstate <= 10'b0011100000;
                        9'b011100000:  nextstate <= 10'b0011110000;
                        9'b011110000:  nextstate <= 10'b0011111000;
                        9'b011111000:  nextstate <= 10'b0011111100;
                        9'b011111100:  nextstate <= 10'b0011111110;
                        9'b011111110:  nextstate <= 10'b0011111111;
                        9'b011111111:  nextstate <= 10'b0111111110;
                        9'b111111110:  nextstate <= 10'b0111111100;
                        9'b111111100:  nextstate <= 10'b0111111000;
                        9'b111111000:  nextstate <= 10'b0111110000;
                        9'b111110000:  nextstate <= 10'b0111100000;
                        9'b111100000:  nextstate <= 10'b0111000000;
                        9'b111000000:  nextstate <= 10'b0110000000;
                        9'b110000000:  nextstate <= 10'b0010000000;
                        default        nextstate <= 10'b0010000000;
                endcase
                state <= state03;
        end
8'b0000_1000:
        begin
                case (state[8:0])
                        9'b000000000:  nextstate <= 10'b0010000000;
                        9'b010000000:  nextstate <= 10'b0011000000;
                        9'b011000000:  nextstate <= 10'b0011100000;
                        9'b011100000:  nextstate <= 10'b0011110000;
                        9'b011110000:  nextstate <= 10'b0011111000;
                        9'b011111000:  nextstate <= 10'b0011111100;
                        9'b011111100:  nextstate <= 10'b0011111110;
                        9'b011111110:  nextstate <= 10'b0011111111;
                        9'b011111111:  nextstate <= 10'b0111111110;
                        9'b111111110:  nextstate <= 10'b0111111100;
                        9'b111111100:  nextstate <= 10'b0111111000;
```

```
                        9'b111111000:  nextstate <= 10'b0111110000;
                        9'b111110000:  nextstate <= 10'b0111100000;
                        9'b111100000:  nextstate <= 10'b0111000000;
                        9'b111000000:  nextstate <= 10'b0110000000;
                        9'b110000000:  nextstate <= 10'b0100000000;
                        9'b100000000:  nextstate <= 10'b0000000001;
                        9'b000000001:  nextstate <= 10'b0000000011;
                        9'b000000011:  nextstate <= 10'b0000000111;
                        9'b000000111:  nextstate <= 10'b0000001111;
                        9'b000001111:  nextstate <= 10'b0000011111;
                        9'b000011111:  nextstate <= 10'b0000111111;
                        9'b000111111:  nextstate <= 10'b0001111111;
                        9'b001111111:  nextstate <= 10'b0111111111;
                        9'b111111111:  nextstate <= 10'b0101111111;
                        9'b101111111:  nextstate <= 10'b0100111111;
                        9'b101111111:  nextstate <= 10'b0100111111;
                        9'b100111111:  nextstate <= 10'b0100011111;
                        9'b100011111:  nextstate <= 10'b0100001111;
                        9'b100001111:  nextstate <= 10'b0100000111;
                        9'b100000111:  nextstate <= 10'b0100000011;
                        9'b100000011:  nextstate <= 10'b0100000001;
                        9'b100000001:  nextstate <= 10'b0000000000;
                        default        nextstate <= 10'b0000000000;
                endcase
                state <= state03;
        end
8'b0000_1001:
        begin
                case (state[8:0])
                        9'b000000001:  nextstate <= 10'b0000000011;
                        9'b000000011:  nextstate <= 10'b0000000111;
                        9'b000000111:  nextstate <= 10'b0000001111;
                        9'b000001111:  nextstate <= 10'b0000011111;
                        9'b000011111:  nextstate <= 10'b0000111111;
                        9'b000111111:  nextstate <= 10'b0001111111;
                        9'b001111111:  nextstate <= 10'b0011111111;
                        9'b011111111:  nextstate <= 10'b0101111111;
                        9'b101111111:  nextstate <= 10'b0100111111;
                        9'b101111111:  nextstate <= 10'b0100111111;
                        9'b100111111:  nextstate <= 10'b0100011111;
                        9'b100011111:  nextstate <= 10'b0100001111;
                        9'b100001111:  nextstate <= 10'b0100000111;
                        9'b100000111:  nextstate <= 10'b0100000011;
                        9'b100000011:  nextstate <= 10'b0000000001;
                        default        nextstate <= 10'b0000000001;
                endcase
                state <= state03;
        end
8'b0001_0000:
        begin
                case (state [7:0])
                        8'b10000000:   nextstate[7:0]<= 8'b01000000;
                        8'b01000000:   nextstate[7:0]<= 8'b00100000;
                        8'b00100000:   nextstate[7:0]<= 8'b00010000;
                        8'b00010000:   nextstate[7:0]<= 8'b00001000;
                        8'b00001000:   nextstate[7:0]<= 8'b00000100;
                        8'b00000100:   nextstate[7:0]<= 8'b00000010;
                        8'b00000010:   nextstate[7:0]<= 8'b00000001;
                        8'b00000001:   nextstate[7:0]<= 8'b10000000;
                        default:       nextstate[7:0]<= 8'b10000000;
                endcase
                state <=state01;
        end
8'b0001_0001:
        begin
                case (state[7:0])
                        8'b10001000:   nextstate[7:0]<= 8'b01000100;
                        8'b01000100:   nextstate[7:0]<= 8'b00100010;
                        8'b00100010:   nextstate[7:0]<= 8'b00010001;
                        8'b00010001:   nextstate[7:0]<= 8'b10001000;
                        default:       nextstate[7:0]<= 8'b10001000;
                endcase
                state <=state01;
        end
8'b0001_0010:
        begin
                case (state)
                        10'b0011000000:       nextstate <= 10'b0001100000;
                        10'b0001100000:       nextstate <= 10'b0000110000;
                        10'b0000110000:       nextstate <= 10'b0000011000;
                        10'b0000011000:       nextstate <= 10'b0000001100;
```

```
                        10'b0000001100:         nextstate <= 10'b0000000110;
                        10'b0000000110:         nextstate <= 10'b0000000011;
                        10'b0000000011:         nextstate <= 10'b0010000001;
                        10'b0010000001:         nextstate <= 10'b0011000000;
                        default:                nextstate <= 10'b0011000000;
                endcase
                state <= state01;
        end
8'b0001_0011:
        begin
                case (state[7:0])
                        8'b11001100:    nextstate[7:0]<= 8'b01100110;
                        8'b01100110:    nextstate[7:0]<= 8'b00110011;
                        8'b00110011:    nextstate[7:0]<= 8'b10011001;
                        8'b10011001:    nextstate[7:0]<= 8'b11001100;
                        default:        nextstate[7:0]<= 8'b11001100;
                endcase
                state <=state01;
        end
8'b0001_0100:
        begin
                case (state)
                        10'b0011100000:         nextstate <= 10'b0001110000;
                        10'b0001110000:         nextstate <= 10'b0000111000;
                        10'b0000111000:         nextstate <= 10'b0000011100;
                        10'b0000011100:         nextstate <= 10'b0000001110;
                        10'b0000001110:         nextstate <= 10'b0000000111;
                        10'b0000000111:         nextstate <= 10'b0010000011;
                        10'b0010000011:         nextstate <= 10'b0011000001;
                        10'b0011000001:         nextstate <= 10'b0011100000;
                        default:                nextstate <= 10'b0011100000;
                endcase
                state <= state01;
        end
8'b0001_0101:
        begin
                case (state[7:0])
                        8'b11101110:    nextstate[7:0]<= 8'b01110111;
                        8'b01110111:    nextstate[7:0]<= 8'b10111011;
                        8'b10111011:    nextstate[7:0]<= 8'b11011101;
                        8'b11011101:    nextstate[7:0]<= 8'b11101110;
                        default:        nextstate[7:0]<= 8'b11101110;
                endcase
                state <=state01;
        end
8'b0001_0110:
        begin
                case (state)
                        10'b0011110000:         nextstate <= 10'b0001111000;
                        10'b0001111000:         nextstate <= 10'b0000111100;
                        10'b0000111100:         nextstate <= 10'b0000011110;
                        10'b0000011110:         nextstate <= 10'b0000001111;
                        10'b0000001111:         nextstate <= 10'b0010000111;
                        10'b0010000111:         nextstate <= 10'b0011000011;
                        10'b0011000011:         nextstate <= 10'b0011100001;
                        10'b0011100001:         nextstate <= 10'b0011110000;
                        default:                nextstate <= 10'b0011110000;
                endcase
                state <= state01;
        end
8'b0001_0111:
        begin
                case (state)
                        10'b0011111000:         nextstate <= 10'b0001111100;
                        10'b0001111100:         nextstate <= 10'b0000111110;
                        10'b0000111110:         nextstate <= 10'b0000011111;
                        10'b0000011111:         nextstate <= 10'b0010001111;
                        10'b0010001111:         nextstate <= 10'b0011000111;
                        10'b0011000111:         nextstate <= 10'b0011100011;
                        10'b0011100011:         nextstate <= 10'b0011110001;
                        10'b0011110001:         nextstate <= 10'b0011111000;
                        default:                nextstate <= 10'b0011111000;
                endcase
                state <= state01;
        end
8'b0001_1000:
        begin
                case (state)
                        10'b0011111100:         nextstate <= 10'b0001111110;
                        10'b0001111110:         nextstate <= 10'b0000111111;
                        10'b0000111111:         nextstate <= 10'b0010011111;
```

```
                        10'b0010011111:         nextstate <= 10'b0011001111;
                        10'b0011001111:         nextstate <= 10'b0011100111;
                        10'b0011100111:         nextstate <= 10'b0011110011;
                        10'b0011110011:         nextstate <= 10'b0011111001;
                        10'b0011111001:         nextstate <= 10'b0011111100;
                        default:                nextstate <= 10'b0011111100;
                endcase
                state <= state01;
        end
8'b0001_1001:
        begin
                case (state)
                        10'b0011111110:         nextstate <= 10'b0001111111;
                        10'b0001111111:         nextstate <= 10'b0010111111;
                        10'b0010111111:         nextstate <= 10'b0011011111;
                        10'b0011011111:         nextstate <= 10'b0011101111;
                        10'b0011101111:         nextstate <= 10'b0011110111;
                        10'b0011110111:         nextstate <= 10'b0011111011;
                        10'b0011111011:         nextstate <= 10'b0011111101;
                        10'b0011111101:         nextstate <= 10'b0011111110;
                        default:                nextstate <= 10'b0011111110;
                endcase
                state <= state01;
        end
8'b0010_0000:
        begin
                nextstate <= 0;
                if (rand[2:0]==0) nextstate[0]<=1;
                if (rand[2:0]==1) nextstate[1]<=1;
                if (rand[2:0]==2) nextstate[2]<=1;
                if (rand[2:0]==3) nextstate[3]<=1;
                if (rand[2:0]==4) nextstate[4]<=1;
                if (rand[2:0]==5) nextstate[5]<=1;
                if (rand[2:0]==6) nextstate[6]<=1;
                if (rand[2:0]==7) nextstate[7]<=1;
                state <= state01;
        end
8'b0010_0001:
        begin
                nextstate <= 0;
                if ((rand[2:0]==0)|(rand[5:3]==0)) nextstate[0]<=1;
                if ((rand[2:0]==1)|(rand[5:3]==1)) nextstate[1]<=1;
                if ((rand[2:0]==2)|(rand[5:3]==2)) nextstate[2]<=1;
                if ((rand[2:0]==3)|(rand[5:3]==3)) nextstate[3]<=1;
                if ((rand[2:0]==4)|(rand[5:3]==4)) nextstate[4]<=1;
                if ((rand[2:0]==5)|(rand[5:3]==5)) nextstate[5]<=1;
                if ((rand[2:0]==6)|(rand[5:3]==6)) nextstate[6]<=1;
                if ((rand[2:0]==7)|(rand[5:3]==7)) nextstate[7]<=1;
                state <= state01;
        end
8'b0010_0010:
        begin
                nextstate <= 0;
                if ((rand[2:0]==0)|(rand[5:3]==0)|(rand[8:6]==0)) nextstate[0]<=1;
                if ((rand[2:0]==1)|(rand[5:3]==1)|(rand[8:6]==1)) nextstate[1]<=1;
                if ((rand[2:0]==2)|(rand[5:3]==2)|(rand[8:6]==2)) nextstate[2]<=1;
                if ((rand[2:0]==3)|(rand[5:3]==3)|(rand[8:6]==3)) nextstate[3]<=1;
                if ((rand[2:0]==4)|(rand[5:3]==4)|(rand[8:6]==4)) nextstate[4]<=1;
                if ((rand[2:0]==5)|(rand[5:3]==5)|(rand[8:6]==5)) nextstate[5]<=1;
                if ((rand[2:0]==6)|(rand[5:3]==6)|(rand[8:6]==6)) nextstate[6]<=1;
                if ((rand[2:0]==7)|(rand[5:3]==7)|(rand[8:6]==7)) nextstate[7]<=1;
                state <= state01;
        end
8'b0010_0011:
        begin
                nextstate <= 0;
                if (rand[1:0]==0) nextstate[1:0]<=3;
                if (rand[1:0]==1) nextstate[3:2]<=3;
                if (rand[1:0]==2) nextstate[5:4]<=3;
                if (rand[1:0]==3) nextstate[7:6]<=3;
                state <= state01;
        end
8'b0010_0100:
        begin
                nextstate <= 0;
                if (rand[0]==0) nextstate[3:0]<=15;
                if (rand[0]==1) nextstate[7:4]<=15;
                state <= state04;
        end
8'b0010_0101:
        begin
```

```
                nextstate <= 0;
                if ((randn1==0)|(randn2==0)) nextstate[0]<=1;
                if ((randn1==1)|(randn2==1)) nextstate[1]<=1;
                if ((randn1==2)|(randn2==2)) nextstate[2]<=1;
                if ((randn1==3)|(randn2==3)) nextstate[3]<=1;
                if ((randn1==4)|(randn2==4)) nextstate[4]<=1;
                if ((randn1==5)|(randn2==5)) nextstate[5]<=1;
                if ((randn1==6)|(randn2==6)) nextstate[6]<=1;
                if ((randn1==7)|(randn2==7)) nextstate[7]<=1;
                state <= state02;
        end
8'b0010_0110:
        begin
                nextstate <= 0;
                if ((randn1==0)|(randn2==0)|(randn3==0)) nextstate[0]<=1;
                if ((randn1==1)|(randn2==1)|(randn3==1)) nextstate[1]<=1;
                if ((randn1==2)|(randn2==2)|(randn3==2)) nextstate[2]<=1;
                if ((randn1==3)|(randn2==3)|(randn3==3)) nextstate[3]<=1;
                if ((randn1==4)|(randn2==4)|(randn3==4)) nextstate[4]<=1;
                if ((randn1==5)|(randn2==5)|(randn3==5)) nextstate[5]<=1;
                if ((randn1==6)|(randn2==6)|(randn3==6)) nextstate[6]<=1;
                if ((randn1==7)|(randn2==7)|(randn3==7)) nextstate[7]<=1;
                state <= state02;
        end
8'b0010_0111:
        begin
                nextstate <= 0;
                if ((rand[2:0]==0)) begin nextstate[0]<=1;nextstate[1]<=1; end
                if ((rand[2:0]==1)) begin nextstate[1]<=1;nextstate[2]<=1; end
                if ((rand[2:0]==2)) begin nextstate[2]<=1;nextstate[3]<=1; end
                if ((rand[2:0]==3)) begin nextstate[3]<=1;nextstate[4]<=1; end
                if ((rand[2:0]==4)) begin nextstate[4]<=1;nextstate[5]<=1; end
                if ((rand[2:0]==5)) begin nextstate[5]<=1;nextstate[6]<=1; end
                if ((rand[2:0]==6)) begin nextstate[6]<=1;nextstate[7]<=1; end
                if ((rand[2:0]==7)) begin nextstate[7]<=1;nextstate[0]<=1; end
                state <= state02;
        end
8'b0010_1000:
        begin
                nextstate <= 0;
                if ((rand[2:0]==0)|(rand[5:3]==0))
                        begin
                                nextstate[0]<=1;
                                nextstate[1]<=1;
                        end
                if ((rand[2:0]==1)|(rand[5:3]==1))
                        begin
                                nextstate[1]<=1;
                                nextstate[2]<=1;
                        end
                if ((rand[2:0]==2)|(rand[5:3]==2))
                        begin
                                nextstate[2]<=1;
                                nextstate[3]<=1;
                        end
                if ((rand[2:0]==3)|(rand[5:3]==3))
                        begin
                                nextstate[3]<=1;
                                nextstate[4]<=1;
                        end
                if ((rand[2:0]==4)|(rand[5:3]==4))
                        begin
                                nextstate[4]<=1;
                                nextstate[5]<=1;
                        end
                if ((rand[2:0]==5)|(rand[5:3]==5))
                        begin
                                nextstate[5]<=1;
                                nextstate[6]<=1;
                        end
                if ((rand[2:0]==6)|(rand[5:3]==6))
                        begin
                                nextstate[6]<=1;
                                nextstate[7]<=1;
                        end
                if ((rand[2:0]==7)|(rand[5:3]==7))
                        begin
                                nextstate[7]<=1;
                                nextstate[0]<=1;
                        end
                state <= state02;
```

```
        end
8'b0010_1001:
      begin
                    nextstate <= 0;
            if ((rand[2:0]==0)|(rand[3:1]==0)|(rand[4:2]==0)) nextstate[2:0]<=7;
            if ((rand[2:0]==1)|(rand[3:1]==1)|(rand[4:2]==1)) nextstate[3:1]<=7;
            if ((rand[2:0]==2)|(rand[3:1]==2)|(rand[4:2]==2)) nextstate[4:2]<=7;
            if ((rand[2:0]==3)|(rand[3:1]==3)|(rand[4:2]==3)) nextstate[5:3]<=7;
            if ((rand[2:0]==4)|(rand[3:1]==4)|(rand[4:2]==4)) nextstate[6:4]<=7;
            if ((rand[2:0]==5)|(rand[3:1]==5)|(rand[4:2]==5)) nextstate[7:5]<=7;
            if ((rand[2:0]==6)|(rand[3:1]==6)|(rand[4:2]==6))
                begin
                        nextstate[7:6]<=3;
                        nextstate[0]<=1;
                end
            if ((rand[2:0]==7)|(rand[3:1]==7)|(rand[4:2]==7))
                begin
                        nextstate[7]<=1;
                        nextstate[1:0]<=3;
                end
           state <= state03;
        end
8'b0011_0000:  state <=state04;
8'b0011_0001:
        begin
                case (state[7:0])
                        8'b00000000:   nextstate[7:0]<= 8'b11111111;
                        8'b11111111:   nextstate[7:0]<= 8'b00000000;
                        default:       nextstate[7:0]<= 8'b00000000;
                endcase
                state <=state01;
        end
8'b0011_0010: state <= state04;
8'b0011_0011: state <= {ChannelIn[7], ChannelIn[5], ChannelIn[3], ChannelIn[1],
ChannelIn[0], ChannelIn[2], ChannelIn[4], ChannelIn[6]};
8'b0011_0100: state <= state05;
8'b0011_0101: state <= state06;
8'b0011_0110:
        begin
                case (state[7:0])
                        8'b01010101:   nextstate[7:0]<= 8'b10101010;
                        8'b10101010:   nextstate[7:0]<= 8'b01010101;
                        default:       nextstate[7:0]<= 8'b01010101;
                endcase
                state <=state01;
        end
8'b0011_0111:
        begin
                case (state)
                        10'b0000001110:        nextstate <= 10'b0001010100;
                        10'b0001010100:        nextstate <= 10'b0001001010;
                        10'b0001001010:        nextstate <= 10'b0000010001;
                        10'b0000010001:        nextstate <= 10'b0010100100;
                        10'b0010100100:        nextstate <= 10'b0101010100;
                        10'b0101010100:        nextstate <= 10'b0011100000;
                        10'b0011100000:        nextstate <= 10'b0000000001;
                        10'b0000000001:        nextstate <= 10'b0000001110;
                        default:               nextstate <= 10'b0000001110;
                endcase
                state <=state01;
        end
8'b0011_1000:
        begin
                case (state)
                        10'b0000000110:        nextstate <= 10'b0000010100;
                        10'b0000010100:        nextstate <= 10'b0001001000;
                        10'b0001001000:        nextstate <= 10'b0000010001;
                        10'b0000010001:        nextstate <= 10'b0000100100;
                        10'b0000100100:        nextstate <= 10'b0001010000;
                        10'b0001010000:        nextstate <= 10'b0011000000;
                        10'b0011000000:        nextstate <= 10'b0000000001;
                        10'b0000000001:        nextstate <= 10'b0000000110;
                        default:               nextstate <= 10'b0000000110;
                endcase
                state <=state01;
        end
8'b0011_1001:
        begin
                case (state)
                        10'b0000011110:        nextstate <= 10'b0001010101;
                        10'b0001010101:        nextstate <= 10'b0001011010;
```

```
                        10'b0001011010:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0010110100;
                        10'b0010110100:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011110000;
                        10'b0011110000:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0000011110;
                        default:                nextstate <= 10'b0000011110;
                endcase
                state <=state01;
        end
8'b0100_0000:
        begin
                case (state)
                        10'b0000111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0011011010;
                        10'b0011011010:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0010110110;
                        10'b0010110110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011111000;
                        10'b0011111000:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0000111110;
                        default:                nextstate <= 10'b0000111110;
                endcase
                state <=state01;
        end
8'b0100_0001:
        begin
                case (state)
                        10'b0001111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0011011110;
                        10'b0011011110:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0011110110;
                        10'b0011110110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011111100;
                        10'b0011111100:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0001111110;
                        default:                nextstate <= 10'b0001111110;
                endcase
                state <=state01;
        end
8'b0100_0010:
        begin
                case (state)
                        10'b0011111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0111111110;
                        10'b0111111110:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b1011111110;
                        10'b1011111110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b1111111110;
                        10'b1111111110:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0011111110;
                        default:                nextstate <= 10'b0011111110;
                endcase
                state <=state01;
        end
8'b0100_0011:
        begin
                case (state)
                        10'b0011111111:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0111111111;
                        10'b0111111111:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b1011111111;
                        10'b1011111111:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b1111111111;
                        10'b1111111111:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0011111111;
                        default:                nextstate <= 10'b0011111111;
                endcase
                state <=state01;
        end
8'b0100_0100:
        begin
                case (state)
                        10'b0000000110:         nextstate <= 10'b0000010100;
                        10'b0000010100:         nextstate <= 10'b0001001000;
                        10'b0001001000:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0000100100;
                        10'b0000100100:         nextstate <= 10'b0001010000;
                        10'b0001010000:         nextstate <= 10'b0011000000;
                        10'b0011000000:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0000000110;
```

```
                        default:                nextstate <= 10'b0000000110;
                endcase
                state <=state07;
        end
8'b0100_0101:
        begin
                case (state)
                        10'b0000011110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0001011010;
                        10'b0001011010:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0010110100;
                        10'b0010110100:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011110000;
                        10'b0011110000:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0000011110;
                        default:                nextstate <= 10'b0000011110;
                endcase
                state <=state07;
        end
8'b0100_0110:
        begin
                case (state)
                        10'b0000111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0011011010;
                        10'b0011011010:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0010110110;
                        10'b0010110110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011111000;
                        10'b0011111000:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0000111110;
                        default:                nextstate <= 10'b0000111110;
                endcase
                state <=state07;
        end
8'b0100_0111:
        begin
                case (state)
                        10'b0001111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0011011110;
                        10'b0011011110:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b0011110110;
                        10'b0011110110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b0011111100;
                        10'b0011111100:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0001111110;
                        default:                nextstate <= 10'b0001111110;
                endcase
                state <=state07;
        end
8'b0100_1000:
        begin
                case (state)
                        10'b0011111110:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0111111110;
                        10'b0111111110:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b1011111110;
                        10'b1011111110:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b1111111110;
                        10'b1111111110:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0011111110;
                        default:                nextstate <= 10'b0011111110;
                endcase
                state <=state07;
        end
8'b0100_1001:
        begin
                case (state)
                        10'b0011111111:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0111111111;
                        10'b0111111111:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b1011111111;
                        10'b1011111111:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b1111111111;
                        10'b1111111111:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0011111111;
                        default:                nextstate <= 10'b0011111111;
                endcase
                state <=state07;
        end
8'b0101_0000: state[7:0] <= ~ChannelIn;
8'b0101_0001:
```

```
        begin
                case (state[7:0])
                        8'b11100111:   nextstate[7:0]<= 8'b11011011;
                        8'b11011011:   nextstate[7:0]<= 8'b10111101;
                        8'b10111101:   nextstate[7:0]<= 8'b01111110;
                        8'b01111110:   nextstate[7:0]<= 8'b11100111;
                        default:       nextstate[7:0]<= 8'b11100111;
                endcase
                state          <= state01;
        end
8'b0101_0010:
        begin
                case (state [7:0])
                        8'b11100111:   nextstate[7:0]<= 8'b01111110;
                        8'b11011011:   nextstate[7:0]<= 8'b11100111;
                        8'b10111101:   nextstate[7:0]<= 8'b11011011;
                        8'b01111110:   nextstate[7:0]<= 8'b10111101;
                        default:       nextstate[7:0]<= 8'b01111110;
                endcase
                state <= state01;
        end
8'b0101_0011:
        begin
                case (state[8:0])
                        9'b001111110:  nextstate <= 10'b0010111101;
                        9'b010111101:  nextstate <= 10'b0011011011;
                        9'b011011011:  nextstate <= 10'b0011100111;
                        9'b011100111:  nextstate <= 10'b0111011011;
                        9'b111011011:  nextstate <= 10'b0110111101;
                        9'b110111101:  nextstate <= 10'b0001111110;
                        default:       nextstate <= 10'b0001111110;
                endcase
                state <= state01;
        end
8'b0101_0100:
        begin
                case (state[8:0])
                        9'b001111110:  nextstate <= 10'b0010111101;
                        9'b010111101:  nextstate <= 10'b0011011011;
                        9'b011011011:  nextstate <= 10'b0011100111;
                        9'b011100111:  nextstate <= 10'b0111100111;
                        9'b111100111:  nextstate <= 10'b0111011011;
                        9'b111011011:  nextstate <= 10'b0110111101;
                        9'b110111101:  nextstate <= 10'b0101111110;
                        9'b101111110:  nextstate <= 10'b0001111110;
                        default:       nextstate <= 10'b0001111110;
                endcase
                state <= state01;
        end
8'b0101_0101:
        begin
                case (state[8:0])
                        9'b011100111:  nextstate <= 10'b0011000011;
                        9'b011000011:  nextstate <= 10'b0010000001;
                        9'b010000001:  nextstate <= 10'b0000000000;
                        9'b000000000:  nextstate <= 10'b0100000000;
                        9'b100000000:  nextstate <= 10'b0110000001;
                        9'b110000001:  nextstate <= 10'b0111000011;
                        9'b111000011:  nextstate <= 10'b0011100111;
                        default        nextstate <= 10'b0011100111;
                endcase
                state <= state03;
        end
8'b0101_0110:
        begin
                case (state[8:0])
                        9'b011111111:  nextstate <= 10'b0011100111;
                        9'b011100111:  nextstate <= 10'b0011000011;
                        9'b011000011:  nextstate <= 10'b0010000001;
                        9'b010000001:  nextstate <= 10'b0000000000;
                        9'b000000000:  nextstate <= 10'b0100000000;
                        9'b100000000:  nextstate <= 10'b0110000001;
                        9'b110000001:  nextstate <= 10'b0111000011;
                        9'b111000011:  nextstate <= 10'b0111100111;
                        9'b111100111:  nextstate <= 10'b0011111111;
                        default        nextstate <= 10'b0011111111;
                endcase
                state <= state03;
        end
8'b0101_0111:
        begin
```

```
                case (state[8:0])
                        9'b001111111:  nextstate <= 10'b0000111111;
                        9'b000111111:  nextstate <= 10'b0000011111;
                        9'b000011111:  nextstate <= 10'b0000001111;
                        9'b000001111:  nextstate <= 10'b0000000111;
                        9'b000000111:  nextstate <= 10'b0000000011;
                        9'b000000011:  nextstate <= 10'b0000000001;
                        9'b000000001:  nextstate <= 10'b0000000000;
                        9'b000000000:  nextstate <= 10'b0100000001;
                        9'b100000001:  nextstate <= 10'b0100000011;
                        9'b100000011:  nextstate <= 10'b0100000111;
                        9'b100000111:  nextstate <= 10'b0100001111;
                        9'b100001111:  nextstate <= 10'b0100011111;
                        9'b100011111:  nextstate <= 10'b0100111111;
                        9'b100111111:  nextstate <= 10'b0101111111;
                        9'b101111111:  nextstate <= 10'b0001111111;
                        default        nextstate <= 10'b0001111111;
                endcase
                state <= state03;
        end
8'b0101_1000:
        begin
                case (state[8:0])
                        9'b011111111:  nextstate <= 10'b0001111111;
                        9'b001111111:  nextstate <= 10'b0000111111;
                        9'b000111111:  nextstate <= 10'b0000011111;
                        9'b000011111:  nextstate <= 10'b0000001111;
                        9'b000001111:  nextstate <= 10'b0000000111;
                        9'b000000111:  nextstate <= 10'b0000000011;
                        9'b000000011:  nextstate <= 10'b0000000001;
                        9'b000000001:  nextstate <= 10'b0000000000;
                        9'b000000000:  nextstate <= 10'b0100000001;
                        9'b100000001:  nextstate <= 10'b0100000011;
                        9'b100000011:  nextstate <= 10'b0100000111;
                        9'b100000111:  nextstate <= 10'b0100001111;
                        9'b100001111:  nextstate <= 10'b0100011111;
                        9'b100011111:  nextstate <= 10'b0100111111;
                        9'b100111111:  nextstate <= 10'b0101111111;
                        9'b101111111:  nextstate <= 10'b0111111111;
                        9'b111111111:  nextstate <= 10'b0011111110;
                        9'b011111110:  nextstate <= 10'b0011111100;
                        9'b011111100:  nextstate <= 10'b0011111000;
                        9'b011111000:  nextstate <= 10'b0011110000;
                        9'b011110000:  nextstate <= 10'b0011100000;
                        9'b011100000:  nextstate <= 10'b0011000000;
                        9'b011000000:  nextstate <= 10'b0010000000;
                        9'b010000000:  nextstate <= 10'b0100000000;
                        9'b100000000:  nextstate <= 10'b0110000000;
                        9'b110000000:  nextstate <= 10'b0111000000;
                        9'b111000000:  nextstate <= 10'b0111100000;
                        9'b111100000:  nextstate <= 10'b0111110000;
                        9'b111110000:  nextstate <= 10'b0111111000;
                        9'b111111000:  nextstate <= 10'b0111111100;
                        9'b111111100:  nextstate <= 10'b0111111110;
                        9'b111111110:  nextstate <= 10'b0011111111;
                        default        nextstate <= 10'b0011111111;
                endcase
                state <= state03;
        end
8'b0101_1001:
        begin
                case (state[8:0])
                        9'b011111110:  nextstate <= 10'b0011111100;
                        9'b011111100:  nextstate <= 10'b0011111000;
                        9'b011111000:  nextstate <= 10'b0011110000;
                        9'b011110000:  nextstate <= 10'b0011100000;
                        9'b011100000:  nextstate <= 10'b0011000000;
                        9'b011000000:  nextstate <= 10'b0010000000;
                        9'b010000000:  nextstate <= 10'b0000000000;
                        9'b000000000:  nextstate <= 10'b0110000000;
                        9'b110000000:  nextstate <= 10'b0111000000;
                        9'b111000000:  nextstate <= 10'b0111100000;
                        9'b111100000:  nextstate <= 10'b0111110000;
                        9'b111110000:  nextstate <= 10'b0111111000;
                        9'b111111000:  nextstate <= 10'b0111111100;
                        9'b111111100:  nextstate <= 10'b0011111110;
                        default        nextstate <= 10'b0011111110;
                endcase
                state <= state03;
        end
8'b0110_0000:
```

```
        begin
                case (state [7:0])
                        8'b10000000:    nextstate[7:0]<= 8'b00000001;
                        8'b01000000:    nextstate[7:0]<= 8'b10000000;
                        8'b00100000:    nextstate[7:0]<= 8'b01000000;
                        8'b00010000:    nextstate[7:0]<= 8'b00100000;
                        8'b00001000:    nextstate[7:0]<= 8'b00010000;
                        8'b00000100:    nextstate[7:0]<= 8'b00001000;
                        8'b00000010:    nextstate[7:0]<= 8'b00000100;
                        8'b00000001:    nextstate[7:0]<= 8'b00000010;
                        default:        nextstate[7:0]<= 8'b10000000;
                endcase
                state <=state01;
        end
8'b0110_0001:
        begin
                case (state[7:0])
                        8'b10001000:    nextstate[7:0]<= 8'b00010001;
                        8'b01000100:    nextstate[7:0]<= 8'b10001000;
                        8'b00100010:    nextstate[7:0]<= 8'b01000100;
                        8'b00010001:    nextstate[7:0]<= 8'b00100010;
                        default:        nextstate[7:0]<= 8'b10001000;
                endcase
                state <=state01;
        end
8'b0110_0010:
        begin
                case (state)
                        10'b0011000000:         nextstate <= 10'b0010000001;
                        10'b0001100000:         nextstate <= 10'b0011000000;
                        10'b0000110000:         nextstate <= 10'b0001100000;
                        10'b0000011000:         nextstate <= 10'b0000110000;
                        10'b0000001100:         nextstate <= 10'b0000011000;
                        10'b0000000110:         nextstate <= 10'b0000001100;
                        10'b0000000011:         nextstate <= 10'b0000000110;
                        10'b0010000001:         nextstate <= 10'b0000000011;
                        default:                nextstate <= 10'b0011000000;
                endcase
                state <= state01;
        end
8'b0110_0011:
        begin
                case (state[7:0])
                        8'b11001100:    nextstate[7:0]<= 8'b10011001;
                        8'b01100110:    nextstate[7:0]<= 8'b11001100;
                        8'b00110011:    nextstate[7:0]<= 8'b01100110;
                        8'b10011001:    nextstate[7:0]<= 8'b00110011;
                        default:        nextstate[7:0]<= 8'b11001100;
                endcase
                state <=state01;
        end
8'b0110_0100:
        begin
                case (state)
                        10'b0011100000:         nextstate <= 10'b0011000001;
                        10'b0001110000:         nextstate <= 10'b0011100000;
                        10'b0000111000:         nextstate <= 10'b0001110000;
                        10'b0000011100:         nextstate <= 10'b0000111000;
                        10'b0000001110:         nextstate <= 10'b0000011100;
                        10'b0000000111:         nextstate <= 10'b0000001110;
                        10'b0010000011:         nextstate <= 10'b0000000111;
                        10'b0011000001:         nextstate <= 10'b0010000011;
                        default:                nextstate <= 10'b0011100000;
                endcase
                state <= state01;
        end
8'b0110_0101:
        begin
                case (state[7:0])
                        8'b11101110:    nextstate[7:0]<= 8'b11011101;
                        8'b01110111:    nextstate[7:0]<= 8'b11101110;
                        8'b10111011:    nextstate[7:0]<= 8'b01110111;
                        8'b11011101:    nextstate[7:0]<= 8'b10111011;
                        default:        nextstate[7:0]<= 8'b11101110;
                endcase
                state <=state01;
        end
8'b0110_0110:
        begin
                case (state)
                        10'b0011110000:         nextstate <= 10'b0011100001;
```

```
                        10'b0001111000:         nextstate <= 10'b0011110000;
                        10'b0000111100:         nextstate <= 10'b0001111000;
                        10'b0000011110:         nextstate <= 10'b0000111100;
                        10'b0000001111:         nextstate <= 10'b0000011110;
                        10'b0010000111:         nextstate <= 10'b0000001111;
                        10'b0011000011:         nextstate <= 10'b0010000111;
                        10'b0011100001:         nextstate <= 10'b0011000011;
                        default:                nextstate <= 10'b0011110000;
                endcase
                state <= state01;
        end
8'b0110_0111:
        begin
                case (state)
                        10'b0011111000:         nextstate <= 10'b0011110001;
                        10'b0001111100:         nextstate <= 10'b0011111000;
                        10'b0000111110:         nextstate <= 10'b0001111100;
                        10'b0000011111:         nextstate <= 10'b0000111110;
                        10'b0010001111:         nextstate <= 10'b0000011111;
                        10'b0011000111:         nextstate <= 10'b0010001111;
                        10'b0011100011:         nextstate <= 10'b0011000111;
                        10'b0011110001:         nextstate <= 10'b0011100011;
                        default:                nextstate <= 10'b0011111000;
                endcase
                state <= state01;
        end
8'b0110_1000:
        begin
                case (state)
                        10'b0011111100:         nextstate <= 10'b0011111001;
                        10'b0001111110:         nextstate <= 10'b0011111100;
                        10'b0000111111:         nextstate <= 10'b0001111110;
                        10'b0010011111:         nextstate <= 10'b0000111111;
                        10'b0011001111:         nextstate <= 10'b0010011111;
                        10'b0011100111:         nextstate <= 10'b0011001111;
                        10'b0011110011:         nextstate <= 10'b0011100111;
                        10'b0011111001:         nextstate <= 10'b0011110011;
                        default:                nextstate <= 10'b0011111100;
                endcase
                state <= state01;
        end
8'b0110_1001:
        begin
                case (state)
                        10'b0011111110:         nextstate <= 10'b0011111101;
                        10'b0001111111:         nextstate <= 10'b0011111110;
                        10'b0010111111:         nextstate <= 10'b0001111111;
                        10'b0011011111:         nextstate <= 10'b0010111111;
                        10'b0011101111:         nextstate <= 10'b0011011111;
                        10'b0011110111:         nextstate <= 10'b0011101111;
                        10'b0011111011:         nextstate <= 10'b0011110111;
                        10'b0011111101:         nextstate <= 10'b0011111011;
                        default:                nextstate <= 10'b0011111110;
                endcase
                state <= state01;
        end
8'b0111_0000:
        begin
                nextstate <= 10'b111111_1111;
                if (rand[2:0]==0) nextstate[0]<=0;
                if (rand[2:0]==1) nextstate[1]<=0;
                if (rand[2:0]==2) nextstate[2]<=0;
                if (rand[2:0]==3) nextstate[3]<=0;
                if (rand[2:0]==4) nextstate[4]<=0;
                if (rand[2:0]==5) nextstate[5]<=0;
                if (rand[2:0]==6) nextstate[6]<=0;
                if (rand[2:0]==7) nextstate[7]<=0;
                state <= state01;
        end
8'b0111_0001:
        begin
                nextstate <= 10'b111111_1111;
                if ((rand[2:0]==0)|(rand[5:3]==0)) nextstate[0]<=0;
                if ((rand[2:0]==1)|(rand[5:3]==1)) nextstate[1]<=0;
                if ((rand[2:0]==2)|(rand[5:3]==2)) nextstate[2]<=0;
                if ((rand[2:0]==3)|(rand[5:3]==3)) nextstate[3]<=0;
                if ((rand[2:0]==4)|(rand[5:3]==4)) nextstate[4]<=0;
                if ((rand[2:0]==5)|(rand[5:3]==5)) nextstate[5]<=0;
                if ((rand[2:0]==6)|(rand[5:3]==6)) nextstate[6]<=0;
                if ((rand[2:0]==7)|(rand[5:3]==7)) nextstate[7]<=0;
                state <= state01;
```

```
        end
8'b0111_0010:
        begin
                nextstate <= 10'b111111_1111;
                if ((rand[2:0]==0)|(rand[5:3]==0)|(rand[8:6]==0)) nextstate[0]<=0;
                if ((rand[2:0]==1)|(rand[5:3]==1)|(rand[8:6]==1)) nextstate[1]<=0;
                if ((rand[2:0]==2)|(rand[5:3]==2)|(rand[8:6]==2)) nextstate[2]<=0;
                if ((rand[2:0]==3)|(rand[5:3]==3)|(rand[8:6]==3)) nextstate[3]<=0;
                if ((rand[2:0]==4)|(rand[5:3]==4)|(rand[8:6]==4)) nextstate[4]<=0;
                if ((rand[2:0]==5)|(rand[5:3]==5)|(rand[8:6]==5)) nextstate[5]<=0;
                if ((rand[2:0]==6)|(rand[5:3]==6)|(rand[8:6]==6)) nextstate[6]<=0;
                if ((rand[2:0]==7)|(rand[5:3]==7)|(rand[8:6]==7)) nextstate[7]<=0;
                state <= state01;
        end
8'b0111_0011:
        begin
                nextstate <= 10'b111111_1111;
                if (rand[1:0]==0) nextstate[1:0]<=0;
                if (rand[1:0]==1) nextstate[3:2]<=0;
                if (rand[1:0]==2) nextstate[5:4]<=0;
                if (rand[1:0]==3) nextstate[7:6]<=0;
                state <= state01;
        end
8'b0111_0100:
        begin
                nextstate <= 10'b111111_1111;
                if (rand[0]==0) nextstate[3:0]<=0;
                if (rand[0]==1) nextstate[7:4]<=0;
                state <= state04;
        end
8'b0111_0101:
        begin
                nextstate <= 10'b111111_1111;
                if ((randn1==0)|(randn2==0)) nextstate[0]<=0;
                if ((randn1==1)|(randn2==1)) nextstate[1]<=0;
                if ((randn1==2)|(randn2==2)) nextstate[2]<=0;
                if ((randn1==3)|(randn2==3)) nextstate[3]<=0;
                if ((randn1==4)|(randn2==4)) nextstate[4]<=0;
                if ((randn1==5)|(randn2==5)) nextstate[5]<=0;
                if ((randn1==6)|(randn2==6)) nextstate[6]<=0;
                if ((randn1==7)|(randn2==7)) nextstate[7]<=0;
                state <= state02;
        end
8'b0111_0110:
        begin
                nextstate <= 10'b111111_1111;
                if ((randn1==0)|(randn2==0)|(randn3==0)) nextstate[0]<=0;
                if ((randn1==1)|(randn2==1)|(randn3==1)) nextstate[1]<=0;
                if ((randn1==2)|(randn2==2)|(randn3==2)) nextstate[2]<=0;
                if ((randn1==3)|(randn2==3)|(randn3==3)) nextstate[3]<=0;
                if ((randn1==4)|(randn2==4)|(randn3==4)) nextstate[4]<=0;
                if ((randn1==5)|(randn2==5)|(randn3==5)) nextstate[5]<=0;
                if ((randn1==6)|(randn2==6)|(randn3==6)) nextstate[6]<=0;
                if ((randn1==7)|(randn2==7)|(randn3==7)) nextstate[7]<=0;
                state <= state02;
        end
8'b0111_0111:
        begin
                nextstate <= 10'b111111_1111;
                if ((rand[2:0]==0)) begin nextstate[0]<=0;nextstate[1]<=0; end
                if ((rand[2:0]==1)) begin nextstate[1]<=0;nextstate[2]<=0; end
                if ((rand[2:0]==2)) begin nextstate[2]<=0;nextstate[3]<=0; end
                if ((rand[2:0]==3)) begin nextstate[3]<=0;nextstate[4]<=0; end
                if ((rand[2:0]==4)) begin nextstate[4]<=0;nextstate[5]<=0; end
                if ((rand[2:0]==5)) begin nextstate[5]<=0;nextstate[6]<=0; end
                if ((rand[2:0]==6)) begin nextstate[6]<=0;nextstate[7]<=0; end
                if ((rand[2:0]==7)) begin nextstate[7]<=0;nextstate[0]<=0; end
                state <= state02;
        end
8'b0111_1000:
        begin
                nextstate <= 10'b111111_1111;
                if ((rand[2:0]==0)|(rand[5:3]==0))
                        begin
                                nextstate[0]<=0;
                                nextstate[1]<=0;
                        end
                if ((rand[2:0]==1)|(rand[5:3]==1))
                        begin
                                nextstate[1]<=0;
                                nextstate[2]<=0;
```

```
                end
            if ((rand[2:0]==2)|(rand[5:3]==2))
                begin
                        nextstate[2]<=0;
                        nextstate[3]<=0;
                end
            if ((rand[2:0]==3)|(rand[5:3]==3))
                begin
                        nextstate[3]<=0;
                        nextstate[4]<=0;
                end
            if ((rand[2:0]==4)|(rand[5:3]==4))
                begin
                        nextstate[4]<=0;
                        nextstate[5]<=0;
                end
            if ((rand[2:0]==5)|(rand[5:3]==5))
                begin
                        nextstate[5]<=0;
                        nextstate[6]<=0;
                end
            if ((rand[2:0]==6)|(rand[5:3]==6))
                begin
                        nextstate[6]<=0;
                        nextstate[7]<=0;
                end
            if ((rand[2:0]==7)|(rand[5:3]==7))
                begin
                        nextstate[7]<=0;
                        nextstate[0]<=0;
                end
            state <= state02;
        end
8'b0111_1001:
        begin
          nextstate <= 10'b111111_1111;
          if ((rand[2:0]==0)|(rand[3:1]==0)|(rand[4:2]==0)) nextstate[2:0]<=0;
          if ((rand[2:0]==1)|(rand[3:1]==1)|(rand[4:2]==1)) nextstate[3:1]<=0;
          if ((rand[2:0]==2)|(rand[3:1]==2)|(rand[4:2]==2)) nextstate[4:2]<=0;
          if ((rand[2:0]==3)|(rand[3:1]==3)|(rand[4:2]==3)) nextstate[5:3]<=0;
          if ((rand[2:0]==4)|(rand[3:1]==4)|(rand[4:2]==4)) nextstate[6:4]<=0;
          if ((rand[2:0]==5)|(rand[3:1]==5)|(rand[4:2]==5)) nextstate[7:5]<=0;
          if ((rand[2:0]==6)|(rand[3:1]==6)|(rand[4:2]==6))
               begin
                       nextstate[7:6]<=0;
                       nextstate[0]<=0;
               end
          if ((rand[2:0]==7)|(rand[3:1]==7)|(rand[4:2]==7))
               begin
                       nextstate[7]<=0;
                       nextstate[1:0]<=0;
               end
          state <= state03;
        end
8'b1000_0000:  state <=state04;
8'b1000_0001:
        begin
                case (state[7:0])
                        8'b11111111:   nextstate[7:0]<= 8'b00000000;
                        8'b00000000:   nextstate[7:0]<= 8'b11111111;
                        default:       nextstate[7:0]<= 8'b11111111;
                endcase
                state <=state01;
        end
8'b1000_0010: state <= state04;
8'b1000_0011: state <= {~ChannelIn[7], ~ChannelIn[5], ~ChannelIn[3], ~ChannelIn[1],
~ChannelIn[0], ~ChannelIn[2], ~ChannelIn[4], ~ChannelIn[6]};
8'b1000_0100: state <= state05;
8'b1000_0101: state <= state06;
8'b1000_0110:
        begin
                case (state[7:0])
                        8'b10101010:   nextstate[7:0]<= 8'b01010101;
                        8'b01010101:   nextstate[7:0]<= 8'b10101010;
                        default:       nextstate[7:0]<= 8'b10101010;
                endcase
                state <=state01;
        end
8'b1000_0111:
        begin
                case (state)
```

```
                        10'b0011110001:         nextstate <= 10'b0010101011;
                        10'b0010101011:         nextstate <= 10'b0010110101;
                        10'b0010110101:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0001011011;
                        10'b0001011011:         nextstate <= 10'b0110101011;
                        10'b0110101011:         nextstate <= 10'b0000011111;
                        10'b0000011111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011110001;
                        default:                nextstate <= 10'b0011110001;
                endcase
                state <=state01;
        end
8'b1000_1000:
        begin
                case (state)
                        10'b0011111001:         nextstate <= 10'b0011101011;
                        10'b0011101011:         nextstate <= 10'b0010110111;
                        10'b0010110111:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0011011011;
                        10'b0011011011:         nextstate <= 10'b0010101111;
                        10'b0010101111:         nextstate <= 10'b0000111111;
                        10'b0000111111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011111001;
                        default:                nextstate <= 10'b0011111001;
                endcase
                state <=state01;
        end
8'b1000_1001:
        begin
                case (state)
                        10'b0011100001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0010100101;
                        10'b0010100101:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0010110100;
                        10'b0001001011:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000001111;
                        10'b0000001111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011100001;
                        default:                nextstate <= 10'b0011100001;
                endcase
                state <=state01;
        end
8'b1001_0000:
        begin
                case (state)
                        10'b0011000001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0000100101;
                        10'b0000100101:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0001001001;
                        10'b0001001001:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000000111;
                        10'b0000000111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011000001;
                        default:                nextstate <= 10'b0011000001;
                endcase
                state <=state01;
        end
8'b1001_0001:
        begin
                case (state)
                        10'b0010000001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0000100001;
                        10'b0000100001:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0000001001;
                        10'b0000001001:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000000011;
                        10'b0000000011:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0010000001;
                        default:                nextstate <= 10'b0010000001;
                endcase
                state <=state01;
        end
8'b1001_0010:
        begin
                case (state)
                        10'b0000000001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0100000001;
                        10'b0100000001:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b1000000001;
                        10'b1000000001:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b1100000001;
```

```
                        10'b1100000001:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0000000001;
                        default:                nextstate <= 10'b0000000001;
                endcase
                state <=state01;
        end
8'b1001_0011:
        begin
                case (state)
                        10'b0011111111:         nextstate <= 10'b0001010101;
                        10'b0001010101:         nextstate <= 10'b0111111111;
                        10'b0111111111:         nextstate <= 10'b0000010001;
                        10'b0000010001:         nextstate <= 10'b1011111111;
                        10'b1011111111:         nextstate <= 10'b0101010101;
                        10'b0101010101:         nextstate <= 10'b1111111111;
                        10'b1111111111:         nextstate <= 10'b0000000001;
                        10'b0000000001:         nextstate <= 10'b0011111111;
                        default:                nextstate <= 10'b0011111111;
                endcase
                state <=state01;
        end
8'b1001_0100:
        begin
                case (state)
                        10'b0011111001:         nextstate <= 10'b0011101011;
                        10'b0011101011:         nextstate <= 10'b0010110111;
                        10'b0010110111:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0011011011;
                        10'b0011011011:         nextstate <= 10'b0010101111;
                        10'b0010101111:         nextstate <= 10'b0000111111;
                        10'b0000111111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011111001;
                        default:                nextstate <= 10'b0011111001;
                endcase
                state <=state07;
        end
8'b1001_0101:
        begin
                case (state)
                        10'b0011100001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0010100101;
                        10'b0010100101:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0010110100;
                        10'b0001001011:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000001111;
                        10'b0000001111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011100001;
                        default:                nextstate <= 10'b0011100001;
                endcase
                state <=state07;
        end
8'b1001_0110:
        begin
                case (state)
                        10'b0011000001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0000100101;
                        10'b0000100101:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0001001001;
                        10'b0001001001:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000000111;
                        10'b0000000111:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0011000001;
                        default:                nextstate <= 10'b0011000001;
                endcase
                state <=state07;
        end
8'b1001_0111:
        begin
                case (state)
                        10'b0010000001:         nextstate <= 10'b0010101010;
                        10'b0010101010:         nextstate <= 10'b0000100001;
                        10'b0000100001:         nextstate <= 10'b0011101110;
                        10'b0011101110:         nextstate <= 10'b0000001001;
                        10'b0000001001:         nextstate <= 10'b0110101010;
                        10'b0110101010:         nextstate <= 10'b0000000011;
                        10'b0000000011:         nextstate <= 10'b0011111110;
                        10'b0011111110:         nextstate <= 10'b0010000001;
                        default:                nextstate <= 10'b0010000001;
                endcase
                state <=state07;
        end
```

```
        8'b1001_1000:
                begin
                        case (state)
                                10'b0000000001:         nextstate <= 10'b0010101010;
                                10'b0010101010:         nextstate <= 10'b0100000001;
                                10'b0100000001:         nextstate <= 10'b0011101110;
                                10'b0011101110:         nextstate <= 10'b1000000001;
                                10'b1000000001:         nextstate <= 10'b0110101010;
                                10'b0110101010:         nextstate <= 10'b1100000001;
                                10'b1100000001:         nextstate <= 10'b0011111110;
                                10'b0011111110:         nextstate <= 10'b0000000001;
                                default:                nextstate <= 10'b0000000001;
                        endcase
                        state <=state07;
                end
        8'b1001_1001:
                begin
                        case (state)
                                10'b0011111111:         nextstate <= 10'b0001010101;
                                10'b0001010101:         nextstate <= 10'b0111111111;
                                10'b0111111111:         nextstate <= 10'b0000010001;
                                10'b0000010001:         nextstate <= 10'b1011111111;
                                10'b1011111111:         nextstate <= 10'b0101010101;
                                10'b0101010101:         nextstate <= 10'b1111111111;
                                10'b1111111111:         nextstate <= 10'b0000000001;
                                10'b0000000001:         nextstate <= 10'b0011111111;
                                default:                nextstate <= 10'b0011111111;
                        endcase
                        state <=state07;
                end
endcase

always @(posedge ChannelIn[5])
            begin
                state01 <= nextstate;
            end

always @(posedge ChannelIn[2])
            begin
                state07 <= nextstate;
            end

always@(posedge ChannelIn[3] or posedge ChannelIn[2] or posedge ChannelIn[1])
        begin
            case(PNum)
                0010_0101:
                   begin
                        randn1 <= rand[2:0];
                        randn2 <= randn1;
                        state02 <= nextstate;
                   end
                0010_0110:
                   begin
                        randn1 <= rand[2:0];
                        randn2 <= randn1;
                        randn3 <= randn2;
                        state02 <= nextstate;
                   end
                0010_0111: state02 <= nextstate;
                0010_1000: state02 <= nextstate;
            endcase
        end

always @(posedge slowerclock)
        begin
            case(PNum)
                0000_0101:
                   begin
                        if (~ChannelIn[5]) repeatoff<=0;
                        case(state)
                            10'b0000011000:
                                if (ChannelIn[5] & ~repeatoff)
                                    begin
                                        state03 <= 10'b0000111100;
                                        repeatoff <=1;
                                    end
                            default:   state03<= nextstate;
                        endcase
                   end
                0000_0110:
                   begin
```

```
                    if (~ChannelIn[5]) repeatoff<=0;
                    case(state)
                        10'b0000000000:
                            if (ChannelIn[5] & ~repeatoff)
                                begin
                                    state03 <= 10'b0000011000;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_0111:
                begin
                    if (~ChannelIn[5]) repeatoff<=0;
                    case(state)
                        10'b0010000000:
                            if (ChannelIn[5] & ~repeatoff)
                                begin
                                    state03 <= 10'b0011000000;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_1000:
                begin
                    if (~ChannelIn[5]) repeatoff<=0;
                    case(state)
                        10'b0000000000:
                            if (ChannelIn[5] & ~repeatoff)
                                begin
                                    state03 <= 10'b0010000000;
                                    repeatoff <=1;
                                end
                        10'b0100000000:
                            if (ChannelIn[5] & ~repeatoff)
                                begin
                                    state03 <= 10'b0000000001;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_1001:
                begin
                    if (~ChannelIn[5]) repeatoff<=0;
                    case(state)
                        10'b0000000001:
                            if (ChannelIn[5] & ~repeatoff)
                                begin
                                    state03 <= 10'b0000000011;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
        endcase
    end

always @(posedge ChannelIn[5] or negedge ChannelIn[5])
    if(ChannelIn[5])
        begin
            case(PNum)
                8'b0010_0100:  state04 <= nextstate;
                8'b0011_0000:  state04 <= 10'b11_1111_1111;
                8'b0011_0010:
                    begin
                        case (state)
                            10'b0000000000:       state04 <= 10'b0000001111;
                            10'b0100000000:       state04 <= 10'b0011110000;
                            default:              state04 <= 10'b0011110000;
                        endcase
                    end
            8'b0111_0100:  state04 <= nextstate;
            8'b1000_0000:  state04 <= 10'b00_0000_0000;
            8'b0011_0010:
                    begin
                            case (state)
                                    10'b0011111111:       state04 <= 10'b0011110000;
                                    10'b0111111111:       state04 <= 10'b0000001111;
                                    default:              state04 <= 10'b0000001111;
```

```
                                endcase
                        end
          endcase
        end
      else
        begin
          case(PNum)
                  8'b0010_0100:  state04 <= 10'b00_0000_0000;
                  8'b0011_0000:  state04 <= 10'b00_0000_0000;
                  8'b0011_0010:
                          begin
                                  case (state)
                                          10'b0000001111:        state04 <= 10'b0100000000;
                                          10'b0011110000:        state04 <= 10'b0000000000;
                                          default:               state04 <= 10'b0000000000;
                                  endcase
                          end
                  8'b0111_0100:  state04 <= 10'b00_1111_1111;
                  8'b1000_0000:  state04 <= 10'b00_1111_1111;
                  8'b1000_0010:
                          begin
                                  case (state)
                                          10'b0011110000:        state04 <= 10'b0111111111;
                                          10'b0000001111:        state04 <= 10'b0011111111;
                                          default:               state04 <= 10'b0011111111;
                                  endcase
                          end
          endcase
        end

always @(posedge ChannelIn[2] or negedge ChannelIn[2])
    if(ChannelIn[2])
      begin
        case(PNum)
                8'b0011_0100:  state05 <= 10'b11_1111_1111;
                8'b1000_0100:  state05 <= 10'b00_0000_0000;
        endcase
      end
    else
      begin
        case(PNum)
                8'b0011_0100:  state05 <= 10'b00_0000_0000;
                8'b1000_0100:  state05 <= 10'b11_1111_1111;
        endcase
      end

always @(posedge ChannelIn[3] or negedge ChannelIn[3])
    if(ChannelIn[3])
      begin
        case(PNum)
                8'b0011_0101:  state06 <= 10'b11_1111_1111;
                8'b1000_0101:  state06 <= 10'b00_0000_0000;
        endcase
      end
    else
      begin
        case(PNum)
                8'b0011_0101:  state06 <= 10'b00_0000_0000;
                8'b1000_0101:  state06 <= 10'b11_1111_1111;
        endcase
      end
endmodule

module lab4(rows, columns, clk, reset, SSout, SSVDDs, sum, PNUM);
    input [3:0] rows;
    output [5:0] columns;
    input clk, reset;
   output [6:0] SSout;
   output [1:0] SSVDDs;
   output [4:0] sum;
   output [7:0] PNUM;

  reg [5:0] nextcolumns;
  wire [3:0] number,onesdigit,tensdigit,tensdisplay,onesdisplay;
  wire slowclock,reallyslowclock,extremelyslowclock;
  wire [31:0] counterout;
  wire digiten, value;
  wire [1:0] SSVDDsbefore, SSVDDsBlinking;
  counter #(32) clockcounter(clk, 1'b1, reset, 32'b0, counterout);
  assign slowclock = counterout[7];
  assign reallyslowclock = counterout[18];
```

```
  assign extremelyslowclock = counterout[26];
  assign PNUM = {tensdigit,onesdigit};
  flop #(1) androws(clk, &rows, arows);

  always@(posedge clk, posedge reset)
   begin
     if(reset)
        nextcolumns<=6'b000111;
     else
        if(arows)
                begin
                        case(columns)
                                6'b001111: nextcolumns<=6'b000111;
                                6'b000111: nextcolumns<=6'b011111;
                                6'b011111: nextcolumns<=6'b011011;
                                6'b011011: nextcolumns<=6'b101111;
                                6'b101111: nextcolumns<=6'b101101;
                                6'b101101: nextcolumns<=6'b111111;
                                6'b111111: nextcolumns<=6'b111110;
                                6'b111110: nextcolumns<=6'b001111;
                                default: nextcolumns<=6'b000111;
                        endcase
                end
     end

  controlunit cu(reallyslowclock,~&rows,digiten);
  flopen #(6) currcol(clk, &rows, nextcolumns, columns);
  crtonumber makenum({columns[3:0],rows},number, value);
  interpretnumber in(clk, reset, number, (digiten|reset)&value, tensdigit, onesdigit, inprogress,
digselect, tensdisplay, onesdisplay);
  lab3_DSL lab3(tensdisplay,onesdisplay,clk,reset,SSout,SSVDDsbefore,sum);
  assign SSVDDsBlinking = extremelyslowclock | SSVDDsbefore;
  assign SSVDDs[1] = inprogress & digselect ? SSVDDsBlinking[1] : SSVDDsbefore[1];
  assign SSVDDs[0] = inprogress & ~digselect ? SSVDDsBlinking[0] : SSVDDsbefore[0];
endmodule

module counter #(parameter WIDTH = 8)
  (input clk,
   input updown,
    input load,
    input [WIDTH-1:0] d,
    output reg [WIDTH-1:0] q);

  wire [WIDTH-1:0] increment, adderout,regin ;
  mux2 #(WIDTH) udmux(-1,1,updown,increment);

  always@(posedge clk)
   begin
     if(load) q<=d;
     else q<=q+increment;
   end
endmodule

module mux2 #(parameter WIDTH = 8)
             (input  [WIDTH-1:0] d0, d1,
              input              s,
              output [WIDTH-1:0] y);

  assign y = s ? d1 : d0;
endmodule

module flop #(parameter WIDTH = 8)
              (input                  clk,
               input      [WIDTH-1:0] d,
               output reg [WIDTH-1:0] q);

  always @(posedge clk)
    q <= d;
endmodule

module controlunit(clk, nandr, digiten);
    input clk;
   input nandr;
    output digiten;

  wire [11:0] counterout;

  counter #(12) repeattimer(clk, 1'b0, ~nandr, 12'b000000001000, counterout);

  assign digiten = nandr&((&counterout[2:0])|(counterout[11]));
endmodule
```

```
module flopen #(parameter WIDTH = 8)
               (input                  clk,
                input                  en,
                input      [WIDTH-1:0] d,
                output reg [WIDTH-1:0] q);

  always @(posedge clk)
    if (en)    q <= d;
endmodule

module crtonumber(in, out, value);
    input [7:0] in;
    output reg [3:0] out;
   output reg value;

  always@( * )
    case(in)
      8'b11100111:
         begin
                out<=14;
                value<=1;
         end
      8'b11101011:
         begin
                out<=7;
                value<=1;
         end
      8'b11101101:
         begin
                out<=4;
                value<=1;
         end
      8'b11101110:
         begin
                out<=1;
                value<=1;
         end
      8'b11010111:
         begin
                out<=0;
                value<=1;
         end
      8'b11011011:
         begin
                        out<=8;
                        value<=1;
         end
      8'b11011101:
         begin
                out<=5;
                value<=1;
         end
      8'b11011110:
         begin
                out<=2;
                value<=1;
         end
      8'b10110111:
         begin
                out<=15;
                value<=1;
         end
      8'b10111011:
         begin
                out<=9;
                value<=1;
         end
      8'b10111101:
         begin
                out<=6;
                value<=1;
         end
      8'b10111110:
         begin
                out<=3;
                value<=1;
         end
      8'b01110111:
         begin
                out<=13;
```

```
                value<=1;
        end
      8'b01111011:
        begin
                out<=12;
                value<=1;
        end
      8'b01111101:
        begin
                out<=11;
                value<=1;
        end
      8'b01111110:
        begin
                out<=10;
                value<=1;
        end
      default:
        begin
                out<=0;
                value<=0;
        end
    endcase
endmodule

module interpretnumber(clk, reset, number, value, tensdigit, onesdigit, inprogress, digselect,
tenstemp, onestemp);
     input clk, reset;
    input [3:0] number;
     input value;
     output [3:0] tensdigit, tenstemp;
     output [3:0] onesdigit, onestemp;
     output reg inprogress;
     output reg digselect;

    reg [3:0] diginput;
    wire [3:0] tenstemp, onestemp;
    reg cancel, confirm;
flopen #(4) tenstemporary(clk,value,cancel?tensdigit:(digselect?diginput:tenstemp),tenstemp);
flopen #(4) onestemporary(clk,value,cancel?onesdigit:(~digselect?diginput:onestemp),onestemp);
flopen #(4) tenspermanent(clk,confirm|reset,reset?4'b0:tenstemp,tensdigit);
flopen #(4) onespermanent(clk,confirm|reset,reset?4'b0:onestemp,onesdigit);

always@( posedge value )
  begin
  if(inprogress) begin
   confirm<=0;
   cancel<=0;
  end
  case(number)
   0: begin
       if (inprogress) begin
                diginput<=0;
                digselect<=0;
       end
       else begin
                digselect<=1;
                diginput<=0;
                inprogress<=1;
       end
     end
   1: begin
       if (inprogress) begin
                diginput<=1;
                digselect<=0;
       end
       else begin
                digselect<=1;
                diginput<=1;
                inprogress<=1;
       end
     end
   2: begin
       if (inprogress) begin
                diginput<=2;
                digselect<=0;
       end
       else begin
                digselect<=1;
                diginput<=2;
                inprogress<=1;
```

```
    end
  end
3: begin
    if (inprogress) begin
            diginput<=3;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=3;
            inprogress<=1;
    end
  end
4: begin
    if (inprogress) begin
            diginput<=4;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=4;
            inprogress<=1;
    end
  end
5: begin
    if (inprogress) begin
            diginput<=5;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=5;
            inprogress<=1;
    end
  end
6: begin
    if (inprogress) begin
            diginput<=6;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=6;
            inprogress<=1;
    end
  end
7: begin
    if (inprogress) begin
            diginput<=7;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=7;
            inprogress<=1;
    end
  end
8: begin
    if (inprogress) begin
            diginput<=8;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=8;
            inprogress<=1;
    end
  end
9: begin
    if (inprogress) begin
            diginput<=9;
            digselect<=0;
    end
    else begin
            digselect<=1;
            diginput<=9;
            inprogress<=1;
    end
  end
10:begin
    if(~inprogress) digselect<=0;
```

```
        if(diginput<9) diginput<=diginput+1;
        else diginput<=0;
        inprogress<=1;
      end
    11:begin
        if(~inprogress) digselect<=0;
        if(diginput>0) diginput<=diginput-1;
        else diginput<=9;
        inprogress<=1;
      end
    12:begin
        inprogress<=0;
        cancel<=1;
      end
    13:begin
        inprogress<=0;
        confirm<=1;
      end
    14:begin
        inprogress<=1;
        digselect<=1;
      end
    15:begin
        inprogress<=1;
        digselect<=0;
      end
    endcase
  end
endmodule

module lab3_DSL(Switches1, Switches2, clk, reset, SSout, SSVDDs,sum);
    input [3:0] Switches1;
    input [3:0] Switches2;
    input clk;
   input reset;
    output [6:0] SSout;
    output [1:0] SSVDDs;
   output [4:0] sum;
wire [8:0] SwitchSwitch;
wire [3:0] SSdrive;
wire [8:0] sumout;
mux2 #(4) switchmux(Switches1, Switches2, SwitchSwitch[8], SSdrive);
sevenseg ssdecoder(SSdrive, SSout);
assign sum = Switches1 + Switches2;
flopr #(9) counterflop(clk, reset, sumout, SwitchSwitch);
assign sumout= SwitchSwitch+1;
assign SSVDDs = {SwitchSwitch[8], ~SwitchSwitch[8]};
endmodule

module sevenseg(s , seg);
   input [03:0] s;
    output [06:0] seg;
    wire [6:0] nseg;

    //logic for G,F,E,D,C,B,A
    assign nseg[0] = s[3]|(s[2]^s[1])|s[2]&s[1]&~s[0];
    assign nseg[1] = (s[3]^s[2])|(s[3]&s[1])|~(s[3]|s[2]|s[1]|s[0]);
    assign nseg[2] = (s[1]&~s[0])|(s[3]&s[2])|((s[3]&~s[2])&~(s[1]^s[0]))|~(s[3]|s[2]|s[1]|s[0]);
    assign nseg[3] =
(s[3]&~s[1])|(~s[3]&~(s[2]^s[1]^s[0]))|s[3]&(s[3]^s[2]^s[1]^s[0])|s[1]&~(s[3]|s[2]|s[0]);
    assign nseg[4] = (s[3]^s[2])|(s[0]&~s[1])|~(s[3]|s[2])&~(s[1]^s[0]);
    assign nseg[5] =
~(s[3]|s[2])|s[2]&~(s[1]&~s[0])&(s[3]^s[2]^s[1]^s[0])|s[3]&~s[2]&~(s[1]&s[0]);
    assign nseg[6] =
(~(s[3]&~s[2])&s[1])|~(s[3]&s[2])&~(s[3]^s[2]^s[1]^s[0])|s[3]&~(s[2]|s[1]|s[0]);
    assign seg = ~nseg;
endmodule

module flopr #(parameter WIDTH = 8)
              (input                  clk, reset,
               input      [WIDTH-1:0] d,
               output reg [WIDTH-1:0] q);

  always @(posedge clk, posedge reset)
    if (reset) q <= 0;
    else       q <= d;
endmodule
```

Appendix B: Light Sequence Descriptions

Numbers	Name	Description	Functional?
00	Straight Throughput	The lights display a direct interpretation of the current frequency inputs	Yes
01-04	Sequenced Explode	The lights start with a pair in the center and move outwards with each successive input on the ChannelIn[5] band.	Yes
05-09	Timed Explode	This set of FSMs was supposed to trigger on a hit from the ChannelIn[5] band, then progress through a series of states featuring lights exploding from left, right or center. They didn't work.	No
10-19	Chase Sequences	Lights or combinations of light move one step right on each input from ChannelIn[5]	Yes
20-24, 27-29	Straight Random	These FSMs light various combinations and numbers of random lights	Yes
25, 26	Progressive Random	This chooses one random number on each spike in ChannelIn[5]. This number will display through 2 or 3 more spikes, and then turn off.	No
30-32, 34-36	Either Or	These FSMs have only two states, but different patterns and different cues for each one.	Yes
33	Shuffled Through	Reorders the outputs from 00 to 7,5,3,1,0,2,4,6	Yes
37-49	Multi-Paced Chase	These FSMs have different numbers of lights chasing each other left at different paces. The resulting show looks pretty sweet	Yes
50-99	Inverted	These FSMs are basically the opposite of the first 50. Where the lights were on, they're off, and where 10-19 chased right, they now chase left.	See above

86/100 FSMs fully functional.

Appendix C: C Code for PIC A/D Conversion

```
/* final.c
Dane Lindblad and Neel Shah
A/D LightBox Conversion Unit

1. Obtain A/D sample.
2. Write to a table.
3. Repeat for channels 1-7.
4. Calculate threshold value.
5. Set channel output values. */

// Use the 18F452 PIC microprocessor
#include <p18f452.h>
#include <math.h>

void main(void);
int read_value(int);

unsigned char channels; // output
unsigned int voltages[8]; // input
unsigned int multipliers[8];
unsigned int i, sum, avg;

void main (void)
{
	TRISA = 0xFF; TRISE = 0xFF;   TRISC = 0; PORTC = 0;
	ADRESH = 0x00; ADRESL = 0x00;
	ADCON1 = 0x80; ADCON0 = 0x81; // right justified, Fosc/32, start channel 0
	multipliers[0] = 1;    multipliers[1] = 1;
	multipliers[2] = 2;    multipliers[3] = 2;
	multipliers[4] = 2;    multipliers[5] = 3;
	multipliers[6] = 5;    multipliers[7] = 5;
	// 0 - low, 7 - high
	while (1)
	{
		channels = 0; // reset values
		sum = 0;
		for (i=0; i<8; i++)
		{
			voltages[i] = read_value(i)*multipliers[i];
			sum += voltages[i];
		}
		avg = sum/8;
		for (i=0; i<8; i++)
		{
			if (voltages[i] > avg)
				channels += pow(2,i); // setting individual bits
		}
		PORTC = channels;
	}
}

int read_value (int i)
{
	unsigned int rval, adhold;
	ADCON0 += 8*i; // channel select bits
	adhold = ADCON0;
	ADCON0 += 4; // set bit 2 to start A/D conversion
	while (ADCON0 != adhold) {rval = 0;}
	rval = ADRESH;
	rval *= 256;
	rval += ADRESL;
	ADCON0 -= 8*i;
	return rval;
}
```