Dane Lindblad
E168B – Hardware/Software Co-design
Professor D.M. Harris
April 27, 2007

Implementation of LightBox 2.0

Useful Information Included:

- Flash GUI for implementation on XUPV2P Web Server
- Client-Side Flash GUI to Server-Side Application Variable Passing
- C++ Implementation of Variable Pass Band Bandpass Filters
- Externally Triggered Finite State Machines

Preface:

This is just a disclaimer so that before you read this you know that my project did NOT actually work. However, there are several things in here that you may find useful. The Verilog portion of this design was verified last semester in MicroP's, and other than the errors documented here the C++ implementation of bandpass filters should work… just not on the XUPV2P quite yet, as I couldn't find a good method of A/D conversion while working solely on the board.

Project Overview:

The LightBox was a project designed by myself and Neel Shah in fall 2006 MicroP's. It took audio input, and coordinated one of 100 (86 functional) lightshows to match the music. It took audio input, merged the left and right channels, added a DC offset, passed it through 8 different analog bandpass filters, passed those through rectifiers, and analyzed the resulting signals to determine whether the Finite State Machines (FSMs) controlling the lightshow should move to the next state or not.

The problems with LightBox 1.0 were primarily in the user interface. Ideally, the user would be able to see a visual representation of the lightshow they were selecting, as

well as be able to select the specific frequency ranges and voltages for triggering of the FSMs. That was what I had hoped to implement on the XUPV2P this semester and what this paper will walk you through.

Similar to v1.0, the design comprised of 3 main parts, the user interface, the filtering and trigger determination, and the FSMs.

User Interface:

When looking for a method of accomplishing all the items I had set forth for the GUI, the thing that came to mind was the Golden Configuration test of the on-board web server. If the interpretation of graphics was left up to another machine, I could focus more energy on adding the features I wanted to add. Flash also takes up far less space in memory than encoding something with similar functions using just html and gifs. I was also inspired by some examples I saw online of spectrum analyzers using ActionScript 3.0. Since I played a lot of internet flash games (some of which store high scores), I knew that client-to-server transfer of information was possible, so I saw writing my GUI in ActionScript as a viable option.

Unfortunately, due to the nature of Flash and other ActionScript programs, the source is not readily perusable, however if you choose to browse through the file it is available at http://www3.hmc.edu/~dlindblad/Source/GUI/FP-GUI-Program(v8).fla (For Flash Professional 8, or Flash 9 Public Alpha). The user interface is written in ActionScript 2.0, as when I started the project I did not have Flash Professional 8, I had Flash MX 2004, which was not ActionScript 3.0 compatible. As a matter of fact, the Flash 9 Public Alpha is the first version of Flash to support ActionScript 3.0, which made debugging the transfer to AS 3.0 near impossible as the search command wasn't yet

working in the Alpha, and many of the variables generated by selecting behaviors for buttons in AS 2.0 were removed[1] from the language in AS 3.0.

As such, the spectrum analyzer portion of the GUI was out, but I substituted for that with some prefab audio spectra of instrument samples that I took from the editing rooms at Pitzer. The audio spectra were generated (and formatted the way I wanted them) by a Matlab file I wrote which is available in Appendix A, or online[2].

The interface for passing variables between the client-side Flash program and any server side programs is a bit sketchy, as Macromedia determined that any more capabilities than it has now would be security threats. The systems that it does have for this purpose are the LoadVars class, which communicates with the server through PHP or ASP scripts, or the FileReference class, which can upload or download files. I had problems getting the FileReference class to load into my GUI at all, so I abandoned that prospect in favor of the PHP scripts[3].

This, as I realized too late, was a mistake, as the pre-built web-server[4] for the Virtex-4 (not what we have, but both have PowerPC cores) didn't have the ability to run PHP scripts. This is a capability that, theoretically could be added to the server, however it's unlikely as that capability hasn't even been added to Odin yet. This left me with a GUI that could theoretically generate *.txt files passing all my variables to my server-side applications.

---

[1] A complete list of these changes is available online at
http://www3.hmc.edu/~dlindblad/Docs/as3cs_migration.pdf

[2] http://www3.hmc.edu/~dlindblad/Source/Spectra_Generation/findspectrum.m.

[3] The PHP scripts that I wrote for this communication process are available in Appendix B, or online at the following addresses:
http://www3.hmc.edu/~dlindblad/Source/PHPfiles/FSMnumFile.php
http://www3.hmc.edu/~dlindblad/Source/PHPfiles/TriggerInfoFile.php
http://www3.hmc.edu/~dlindblad/Source/PHPfiles/GetTrigInfoFile.php

[4]Documentation for the web server is available from Xilinx:
http://direct.xilinx.com/bvdocs/appnotes/xapp434.pdf

Filtering and Trigger Determination:

One of the benefits of moving away from analog design is the ability to have bandpass filters that are customizable on the fly. C++ has no problem reading text files, so reading the variables that would theoretically have been passed by the GUI is a simple operation. The filtration is the interesting part. My filters[5] were designed around examples from the course notes for E59 (Fall 2004), and the textbook for E102[6].

Applying a bandpass filter could be done by taking the Fourier transform, selecting the correct data sets, padding with zeros, and taking the inverse Fourier transform, but as chapter 8 discusses, the output of a filter can be described as the weighted sum of past and future input values, or if we can tolerate some delay, which I can, just of past values[7]. This weighted average is of the form

$$y_1[n] = y[n-K] = \sum_{k=-K}^{K} a_k x[n-K-k]$$

where a_k is of the form

$$a_k = \frac{1}{2\pi}\int_{2\pi} H\left(e^{j\hat{\omega}}\right)e^{jk\hat{\omega}}d\hat{\omega}$$

and $\hat{\omega}$ is defined by

$$\hat{\omega} = \frac{\omega}{f_s}.$$

---

[5] http://www3.hmc.edu/~dlindblad/Source/Filters/Fp-Filters/FP-Filters.cpp

[6] Oppenheim, Alan V., Alan S. Willsky, with Hamid Nawab, "Signals and Systems" Second edition.

[7] As I didn't actually get to test my code, it may still be riddled with bugs. For instance, I just noticed that the calculations my code is actually doing uses the same a_k, but instead defines y as $y[n-K] = a_k x[n-K-k]$, which is just blatantly wrong. Please do not use that section of my code for future work, as it WILL NOT WORK. Instead, take samples every period T_s, and run it through the filter to obtain $y[n\text{-}K]$ as it comes in. You have the time; the clock speed is 300MHz and the highest audio rate you'll be working with is probably somewhere around 44.1kHz, or somewhere around 6800 clock cycles per sample.

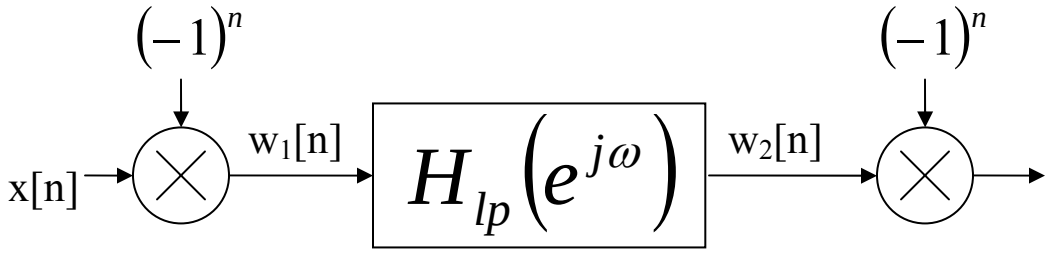
The example that I found useful was the one at the bottom of 8-45 where the coefficients of an ideal lowpass filter are given by

$$a_k = \frac{\hat{\omega}_c}{\pi}\frac{\sin(k\hat{\omega}_c)}{k\hat{\omega}_c} = 2\hat{f}_c\,\mathrm{sinc}\left(2k\hat{f}_c\right)$$

where

$$\hat{\omega}_c = \omega_c T_s = \frac{\omega_c}{f_s}$$
$$\hat{f}_c = \frac{\hat{\omega}_c}{2\pi} = \frac{f_c}{f_s}$$

This gave me an easy way to calculate the filter coefficients for an ideal lowpass filter. However, I needed to make a bandpass filter, so I also needed to make a highpass filter to filter out the lower frequencies. An easy method of making a highpass filter can be found in examples 5.7 and 5.14 in "Signals and Systems." This method consists of applying a frequency shift to a lowpass filter to move the center of the pass band from 0 to π. This can be done simply by adding multipliers both before and after the filter.



Because $(-1)^n = e^{j\pi n}$, when using the frequency shifting property of the Fourier transforms, we find that

$$W_1\left(e^{j\omega}\right) = X\left(e^{j(\omega-\pi)}\right)$$
$$W_2\left(e^{j\omega}\right) = H_{lp}\left(e^{j\omega}\right)X\left(e^{j(\omega-\pi)}\right)$$
$$W_3\left(e^{j\omega}\right) = H_{lp}\left(e^{j(\omega-\pi)}\right)X\left(e^{j(\omega-2\pi)}\right) = H_{lp}\left(e^{j(\omega-\pi)}\right)X\left(e^{j\omega}\right)$$

which is equivalent to sending the signal through a highpass filter. By sending a set of samples through both of these filters, I could filter the incoming audio to find out how much of the signal was in each pass band.

With this information I could determine whether I wanted my trigger signals to be sent or not. The method I chose to decide this was checking for the highest value in that pass band in the last 1023 samples, and determining if that value was above the threshold voltage. If it was, a 1 got shifted in to trigger history array, and if it wasn't a 0 was shifted in. If the trigger history file ever reached all ones or all zeros, the output to the FSMs was changed accordingly by using a combination of bit masks, ANDs, and ORs.

FSMs:

The FSMs used in this project[8] were developed by me last semester, and, other than altering which FSMs got which trigger signals, they remain relatively unchanged. I did alter a few of them if, while making the GUI interpretations of what the lightshow was supposed to do, I found an error that was preventing it from performing like it should. This only really fixed two of the 14 that weren't functional, and more than that were made obsolete by the fact that users can select their own triggers. The Verilog is attached in Appendix D, but it's fairly straightforward so not much explanation is forthcoming.

[8] As I'm already running low on server space on Odin, and there's nothing in the project files you can't get by pasting Appendix D into a new module, I have not made the Verilog available online.

The module consists of an input for the FSM number, an input for the triggers, a giant case statement for selection of the particular FSM, with nested case statements for determination of the next state, and assignment of the output state. This was done in this way because different FSMs accepted different trigger combinations, and hence required different `always` statements. Verilog does not allow the same wire or register to be assigned in different always blocks, so there was one register for each trigger sequence always block, and one register for the FSM case statement to choose from which trigger sequence register to accept values.

Conclusions:

I'm not sure, anything in this will be of use to future research groups/clinic groups/MicroP's Junkies… but if it's anything, it'll probably be the bandpass filtering stuff. Since the books in Blais that look the most suited to this type of thing will most likely be checked out at all times, like they were when I was trying to figure out how to do it, I figure a simple explanation of exactly how to do it would be a good thing to have available somewhere.

This project may have been possible if I had taken the upload/download FileReference route for transferring variables, and had come up with an external means of A/D conversion for the audio. Unfortunately my course load this semester was such that I had to try to complete 60% of the project on the last weekend. With 17 hours of coding on Saturday, and another 19 on Sunday, my fragile mind was just too worn out to even try to think about debugging the project.

Appendix A: Spectra Generation

```
function [f,P] = findspectrum(wavaddress)
[wavname,Fs,bits]=wavread(wavaddress);
y=fft(wavname);
PP=y.*conj(y)/length(wavname);
ff=(0:floor(length(wavname)/2))*(Fs/2)/floor(length(wavname)/2);
indfirst=find(ff>60,1,'first');
indlast =find(ff<16520,1,'last');
P=PP(indfirst:indlast);
f=ff(indfirst:indlast);
plotf = 10.^(log(f)./log(2));
pic = figure('Position', [200 200 520 110]);
colordef black;

semilogx(plotf,P,'LineWidth',2,'Color','g');
axis([min(plotf) max(plotf) min(P) max(P)]);
set(gca,'Units', 'pixels');
set(gca,'Position',[0, 0, 520, 110]);
axis off
```

Appendix B.1: FSMnumFile.php

```
<?php
$FSMnumber  = $_POST['FSMnumber'];
$FSMname    = $_POST['FSMname'];
$add = $FSMnumber . "G" . $FSMname . "G";

$open = fopen('FSMinfo.txt', 'w');

$write = fwrite($open, $add);

fclose($$open);

if($write)
{
 echo "&verify=success&";
}
else
{
 echo "&verify=fail&";
}
?>
```

Appendix B.2: TriggerInfoFile.php

```
<?php
$T1bool  = $_POST['T1bool'];
$T2bool  = $_POST['T2bool'];
$T3bool  = $_POST['T3bool'];
$T1HPF   = $_POST['T1HPF'];
$T2HPF   = $_POST['T2HPF'];
$T3HPF   = $_POST['T3HPF'];
$T1LPF   = $_POST['T1LPF'];
$T2LPF   = $_POST['T2LPF'];
$T3LPF   = $_POST['T3LPF'];
$T1TL    = $_POST['T1TL'];
$T2TL    = $_POST['T2TL'];
$T3TL    = $_POST['T3TL'];

$T1S = $T1bool . "P" . $T1HPF . "P" . $T1LPF . "P" . $T1TL . "P";
$T2S = $T2bool . "P" . $T2HPF . "P" . $T2LPF . "P" . $T2TL . "P";
$T3S = $T3bool . "P" . $T3HPF . "P" . $T3LPF . "P" . $T3TL . "P";

$add = $T1S . $T2S . $T3S;

$open = fopen('TriggerInfo.txt', 'w');

$write = fwrite($open, $add);

fclose($open);

if($write)
{
 echo "&verify=success&";
}
else
{
 echo "&verify=fail&";
}
?>
```

Appendix B.3: GetTrigInfoFile.php

```
<?php

$open = fopen('TriggerInfo.txt', 'r');

$read = fread($open);

$T1bool  = strtok($read, "P");
$T1HPF   = strtok($read, "P");
$T1LPF   = strtok($read, "P");
$T1TL    = strtok($read, "P");
$T2bool  = strtok($read, "P");
$T2HPF   = strtok($read, "P");
$T2LPF   = strtok($read, "P");
$T2TL    = strtok($read, "P");
$T3bool  = strtok($read, "P");
$T3HPF   = strtok($read, "P");
$T3LPF   = strtok($read, "P");
$T3TL    = strtok($read, "P");

$_GET[T1bool]  = $T1bool;
$_GET[T2bool]  = $T2bool;
$_GET[T3bool]  = $T3bool;
$_GET[T1HPF]   = $T1HPF;
$_GET[T2HPF]   = $T2HPF;
$_GET[T3HPF]   = $T3HPF;
$_GET[T1HPF]   = $T1LPF;
$_GET[T2HPF]   = $T2LPF;
$_GET[T3HPF]   = $T3LPF;
$_GET[T1TL]    = $T1TL;
$_GET[T2TL]    = $T2TL;
$_GET[T3TL]    = $T3TL;

if($read)
{
 echo "&verify=success&";
}
else
{
 echo "&verify=fail&";
}
?>
```

Appendix C: Filters and Triggers

```
// FP-Filters.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <algorithm>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "include\xparameters.h"
#include "include\xgpio.h"

#define False false
#define True  true

#define AUDIN       0x7d400000    //Audio In 4 bit device ID GOTTA CHANGE THIS
#define FSM         0x7aa00000
#define OpFreq      300000000     //Operating Frequency 300 MHz
#define FS          44100         //sampling frequency  44.1 kHz
#define Kk          511           //2*Kk+1=N
#define N           1023          //

int Setup()                       //Defines some test files
{
       FILE *fptr;
       fptr = fopen("TriggerInfo.txt", "w");
       fputs ("TrueP71P88P0.9PTrueP244P838P0.8PFalseP1034P1984P0.7P", fptr);
       fclose (fptr);
       fptr = fopen("FSMinfo.txt", "w");
       fputs ("0x35GMulti-Paced 4G", fptr);
       fclose (fptr);
       return 0;


}
typedef struct triggerinfo {      //Holds the Trigger info for 1 trigger
       bool Tbool;
       int    THPF;
       int TLPF;
       double TTL;
}triggerinfo;
typedef struct alltrigs {  //Holds the Trigger info for all triggers
       triggerinfo T1;
       triggerinfo T2;
       triggerinfo T3;
}alltrigs;
alltrigs readdata()        //Reads trigger Data from file
{
       char * buffer;
       triggerinfo T1, T2, T3;
       alltrigs trigs;
       FILE *fptr;
       fptr = fopen("TriggerInfo.txt","r");
       fseek (fptr, 0, SEEK_END);
       buffer = (char*) malloc (sizeof(char)*ftell(fptr));
       fread (buffer, 1, ftell(fptr), fptr);
       char * str = buffer;
```

```
	if(strtok(str, "P")=="True") {T1.Tbool = true;} else {T1.Tbool = false;};
	T1.THPF  = atoi(strtok(NULL, "P"));
	T1.TLPF  = atoi(strtok(NULL, "P"));
	T1.TTL   = strtod(strtok(NULL, "P"),NULL);
	if(strtok(NULL, "P")=="True") {T2.Tbool = true;} else {T2.Tbool =
false;};
	T2.THPF  = atoi(strtok(NULL, "P"));
	T2.TLPF  = atoi(strtok(NULL, "P"));
	T2.TTL   = strtod(strtok(NULL, "P"),NULL);
	if(strtok(NULL, "P")=="True") {T3.Tbool = true;} else {T3.Tbool =
false;};
	T3.THPF  = atoi(strtok(NULL, "P"));
	T3.TLPF  = atoi(strtok(NULL, "P"));
	T3.TTL   = strtod(strtok(NULL, "P"),NULL);
	trigs.T1 = T1;
	trigs.T2 = T2;
	trigs.T3 = T3;
	return trigs;
}
typedef struct FSMinfo		//Holds FSM info
{
	int FSMnum;
	char * FSMname;
}FSMinfo;
int readFSM ()			//Reads FSM info from file
{
	FSMinfo FSMi;
	char * buffer;
	FILE *fptr;
	fptr = fopen("FSMinfo.txt","r");
	fseek (fptr, 0, SEEK_END);
	buffer = (char *) malloc (sizeof(char)*ftell(fptr));
	fread (buffer, 1, ftell(fptr), fptr);
	char * str = buffer;
	FSMi.FSMnum = atoi(strtok(str, "P"));
	FSMi.FSMname = strtok(NULL, "P");
	return FSMi.FSMnum;
}
typedef struct allcoeffs {	//Holds all Filter Coefficients
	double * T1LPcoeffs;
	double * T2LPcoeffs;
	double * T3LPcoeffs;
	double * T1HPcoeffs;
	double * T2HPcoeffs;
	double * T3HPcoeffs;
}allcoeffs;
double* findLPcoeffs(int LPF, int fs, int K)	//Finds Low Pass filter
Coefficients
{
	double a[N];
	double fchat=LPF/fs;
	int k;
	for (k = -K; k<K+1; k++)
	{
		double x = 2*k*fchat;
		if (x != 0) {
			a[-K+k]=2*fchat*sin(x)/x;
		} else {
			a[-K+k]=2*fchat;
		}
	}
	return a;
}
```

```
double* findHPcoeffs(int HPF, int fs, int K)         //Finds High Pass Filter
Coefficients
{
	double a[N];
	double fchat = (fs/2-HPF)/fs;
	int k;
	for (k = -K; k<K+1; k++)
	{
		double x = 2*k*fchat;
		if (x != 0) {
			a[-K+k]=2*fchat*sin(x)/x;
		} else {
			a[-K+k]=2*fchat;
		}
	}
	return a;
}

allcoeffs findAllcoeffs(alltrigs trig, int Fs, int K)//Finds all Filter
Coefficients
{
	allcoeffs coeffs;
	coeffs.T1LPcoeffs = findLPcoeffs(trig.T1.TLPF,Fs,K);
	coeffs.T1HPcoeffs = findHPcoeffs(trig.T1.THPF,Fs,K);
	coeffs.T2LPcoeffs = findLPcoeffs(trig.T2.TLPF,Fs,K);
	coeffs.T2HPcoeffs = findHPcoeffs(trig.T2.THPF,Fs,K);
	coeffs.T3LPcoeffs = findLPcoeffs(trig.T3.TLPF,Fs,K);
	coeffs.T3HPcoeffs = findHPcoeffs(trig.T3.THPF,Fs,K);
	return coeffs;
}
int* sampaudio(int Fs)
{
	XGpio audioin;
	XGpio_Initialize(&audioin, AUDIN);                    //GOTTA CHANGE THIS
	XGpio_SetDataDirection(&audioin, 1, 0xFFFFFFFF);      //GOTTA CHANGE THIS
	int i = 0;
	int values[N];
start:
	time_t start = time(NULL);
keep_going:
	if (difftime(time(NULL),start)>(double) OpFreq/FS)
	{
		if (i >= N)
		{
			return values;
		}
		else
		{
			values[i]=XGpio_DiscreteRead(&audioin,1);
			//GOTTA CHANGE THIS
			++i;
			goto start;
		}
	}
	else
	{
		goto keep_going;
	}
}
double maxValue(double arr[N])
{
	int length = N;
	int max = arr[0];
```

```
	for(int i=1;i<length;i++)
	{
		if(arr[i]>max)
		{
			max = arr[i];
		}
	}
	return max;
}
int _tmain(int argc, _TCHAR* argv[])
{
	int FSMnum;
	int CurrentTrigger;

	Setup();
	int * samples;
	double filtered_samples[3][N];
	int triggerhist[3][6];
	unsigned int tmask[3][2];
	tmask[2][0] = 3; tmask[2][1]=4;
	tmask[1][0] = 5; tmask[1][1]=2;
	tmask[0][0] = 6; tmask[0][1]=1;
	int FSMnumOLD;

	FSMnum = readFSM();
	if (FSMnum != FSMnumOLD)
	{
		FSMnumOLD = FSMnum;
		FP_FSMS_mWriteSlaveReg0(FSM, FSMnum);
	}

reset:
	alltrigs trigs = readdata();
	//Reads Trigger information
	int th = 6;
	int k = 0;

nextset:
	allcoeffs coeffs = findAllcoeffs(trigs, FS, Kk);
	samples = sampaudio(FS);
	for (int i = 0; i<N; i++)
	{
		filtered_samples[0][i]=samples[i]*coeffs.T1LPcoeffs[i]*pow(-1,
(double) i)*coeffs.T1HPcoeffs[i]*pow(-1, (double) i);
		filtered_samples[1][i]=samples[i]*coeffs.T2LPcoeffs[i]*pow(-1,
(double) i)*coeffs.T2HPcoeffs[i]*pow(-1, (double) i);
		filtered_samples[2][i]=samples[i]*coeffs.T3LPcoeffs[i]*pow(-1,
(double) i)*coeffs.T3HPcoeffs[i]*pow(-1, (double) i);
	}

//Checks if trigger is enabled and trigger passes threshold levels
	if (maxValue(filtered_samples[0]) > (double) (trigs.T1.TTL * 0x20000))
	{
		if(trigs.T1.Tbool)
		{
			triggerhist[1][k] = 1;
		} else {
			triggerhist[1][k] = 0;
		}
	}
	if (maxValue(filtered_samples[1]) > (double) (trigs.T2.TTL * 0x20000))
	{
```

```
		if(trigs.T2.Tbool)
		{
			triggerhist[2][k] = 1;
		} else {
			triggerhist[2][k] = 0;
		}
	}
	if (maxValue(filtered_samples[2]) > (double) (trigs.T3.TTL * 0x20000))
	{
		if(trigs.T3.Tbool)
		{
			triggerhist[3][k] = 1;
		} else {
			triggerhist[3][k] = 0;
		}
	}
//Checks if triggers have been steady enough to change values
checkhist:
	int sum[3] = {0,0,0};
	for (int j = 0; j<3; j++)
	{
		for (int i= 0; i<th; i++)
		{
			if (triggerhist[j][i] == 1)
			{
				++sum[j];
			}
		}
		if (sum[j] == 0)
		{
			CurrentTrigger = CurrentTrigger & tmask[j][0];
		}
		else if (sum[j] == th)
		{
			CurrentTrigger = CurrentTrigger | tmask[j][1];
		}
	}

	FP_FSMS_mWriteSlaveReg1(FSM, CurrentTrigger);

	if(k=th-1)
	{
		goto reset;
	}
	else
	{
		goto nextset;
	}
}
```

Appendix D: Verilog FSMS

```
module Main(TriggerIn, LightsOut, clk, PNum, reset);
    input [2:0] TriggerIn;
    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:
        begin
            state[7:0] <= {TriggerIn[2], TriggerIn[2], TriggerIn[2],
                TriggerIn[1], TriggerIn[1], TriggerIn[0], TriggerIn[0], TriggerIn[0]}
        end
    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'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)) nextstate[2:0]<=7;
            if ((rand[2:0]==1)|(rand[3:1]==1)) nextstate[3:1]<=7;
            if ((rand[2:0]==2)|(rand[3:1]==2)) nextstate[4:2]<=7;
            if ((rand[2:0]==3)|(rand[3:1]==3)) nextstate[5:3]<=7;
            if ((rand[2:0]==4)|(rand[3:1]==4)) nextstate[6:4]<=7;
            if ((rand[2:0]==5)|(rand[3:1]==5)) nextstate[7:5]<=7;
            if ((rand[2:0]==6)|(rand[3:1]==6)) begin nextstate[7:6]<=3; nextstate[0]<=1;
end
            if ((rand[2:0]==7)|(rand[3:1]==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 <=
{TriggerIn[2],TriggerIn[1],TriggerIn[2],TriggerIn[0],TriggerIn[0], TriggerIn[1],
TriggerIn[2], TriggerIn[1]};
    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] <= {~TriggerIn[2], ~TriggerIn[2], ~TriggerIn[2],
            ~TriggerIn[1], ~TriggerIn[1], ~TriggerIn[0], ~TriggerIn[0], ~TriggerIn[0]}
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 <=
{~TriggerIn[2],~TriggerIn[1],~TriggerIn[2],~TriggerIn[0],~TriggerIn[0],~TriggerIn[1],~Tri
ggerIn[2],~TriggerIn[1]};
    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 TriggerIn[2] or posedge TriggerIn[1] or posedge TriggerIn[0])
            begin
                state01 <= nextstate;
            end

always @(posedge TriggerIn[2] or posedge TriggerIn[1] or posedge TriggerIn[0])
            begin
                state07 <= nextstate;
            end

always@(posedge TriggerIn[2] or posedge TriggerIn[1] or posedge TriggerIn[0])
        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 (~(TriggerIn[2]|TriggerIn[1]|TriggerIn[0])) repeatoff<=0;
                        case(state)
                            10'b0000011000:
                                if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                    begin
                                        state03 <= 10'b0000111100;
                                        repeatoff <=1;
                                    end
                            default:   state03<= nextstate;
                        endcase
                    end
                0000_0110:
                    begin
                        if (~(TriggerIn[2]|TriggerIn[1]|TriggerIn[0])) repeatoff<=0;
                        case(state)
                            10'b0000000000:
                                if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                    begin
                                        state03 <= 10'b0000011000;
                                        repeatoff <=1;
```

```
                            end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_0111:
                begin
                    if (~(TriggerIn[2]|TriggerIn[1]|TriggerIn[0])) repeatoff<=0;
                    case(state)
                        10'b0010000000:
                            if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                begin
                                    state03 <= 10'b0011000000;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_1000:
                begin
                    if (~(TriggerIn[2]|TriggerIn[1]|TriggerIn[0])) repeatoff<=0;
                    case(state)
                        10'b0000000000:
                            if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                begin
                                    state03 <= 10'b0010000000;
                                    repeatoff <=1;
                                end
                        10'b0100000000:
                            if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                begin
                                    state03 <= 10'b0000000001;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
            0000_1001:
                begin
                    if (~(TriggerIn[2]|TriggerIn[1]|TriggerIn[0])) repeatoff<=0;
                    case(state)
                        10'b0000000001:
                            if ((TriggerIn[2]|TriggerIn[1]|TriggerIn[0]) & ~repeatoff)
                                begin
                                    state03 <= 10'b0000000011;
                                    repeatoff <=1;
                                end
                        default:   state03<= nextstate;
                    endcase
                end
        endcase
    end

always @(posedge (TriggerIn[2]^TriggerIn[1]^TriggerIn[0]) or negedge
(TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
    if((TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
        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 (TriggerIn[2]^TriggerIn[1]^TriggerIn[0]) or negedge
(TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
        if((TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
            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 @((TriggerIn[2]^TriggerIn[1]^TriggerIn[0]) or negedge
(TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
        if((TriggerIn[2]^TriggerIn[1]^TriggerIn[0]))
            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
```