5x5x5 RGB LED Kub

Av: Anders Skoog E08, David Karlsson E08, Stefan Granlund E07

EITF40, VT1 2012

Abstract

The sole purpose of this project is to create an interactive media for displaying three dimensional images and animations. *The result will be a cube of LEDs controlled by a microprocessor interfacing multiple input sources, such as a serial port attached to a computer and an onboard panel of buttons. The LEDs contained in the cube are all 3-colored RGB diodes and the intensity of each diode is controlled by the microprocessor using PWM. The PWM-support enables displaying of a vast amount of different colors and shades, thus providing a robust and advanced platform for animation content.*

Innehåll

Inledning

Syftet med projektet var att bygga en 5x5x5 stor RGB LED kub. Kuben skulle även ha möjligheten att styra färgen på varje enskild lysdiod med hjälp av 8 nivåers pulsbreddsmodulering vilket ger 512 möjliga färger. Kuben skulle även kunna styras dels via tyckknappar på PCB och dels via serieporten från en dator.

Kravspecification

5x5x5 RGB LED kub med 8 nivå PWM på varje LED

Hårdvara:
5x5x5 RGB LED kub (125 LEDs)

Output:

- SPI till 10st 74hc595 shiftreg. som I/O expander för att styra anod på LEDs
- multiplexing av lager med 5 st MOSFET:ar till LED katoderna

Input:

- 6 tryckknappar för inputkommandon
- kommunikation med dator via UART

Mjukvara:

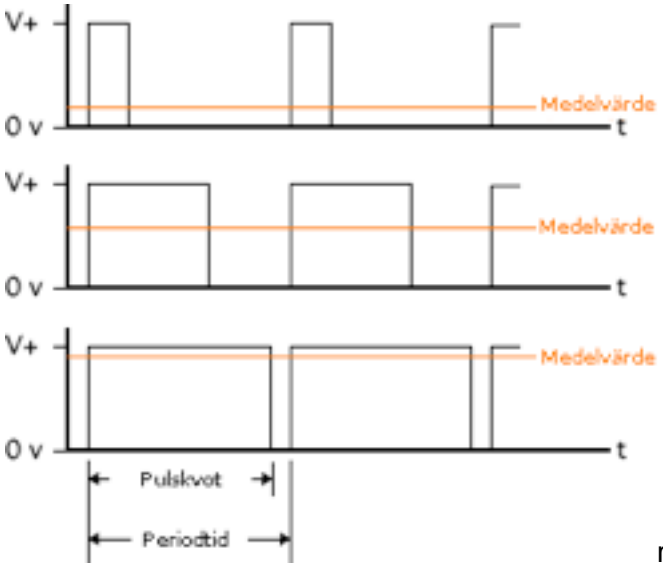
- PWM implementerad i mjukvara i 4 nivåer på alla LED:s
- I mån av tid/processorkraft: 8 nivåer PWM

Funktioner:

- Animationer
- I mån av tid: spela snake ii 3D

Teori

Hur pulsbreddsmodulering (PWM) fungerar

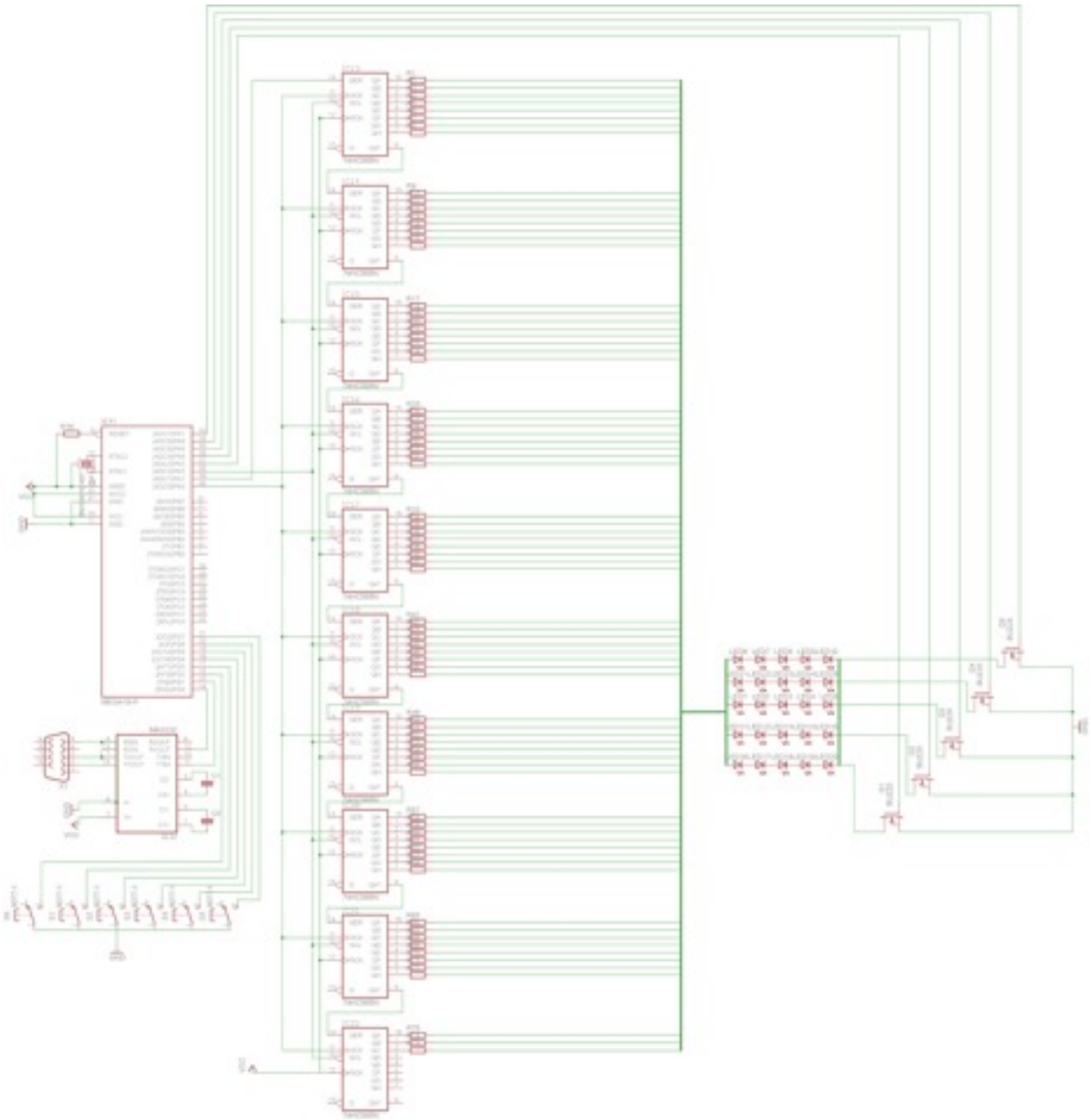


ref [1]

PWM fungerar genom att ändra pulskvoten av en fyrkantsvåg, dvs. ändra hur länge fyrkantsvågen är hög i förhållande till hur länge den är låg vid en fast periodtid. Genom att ändra pulskvoten så kan man ställa medelvärdet på spänningen som kommer ut från en digital I/O pinne och på så sätt skapa en simpel DAC.

[1] http://www.societyofrobots.com/schematics_h-bridgedes.shtml

Kretsbeskrivning



Atmel ATMega16

ATMega16 är en 8bitars mikrokontroller får Atmel. ATMega16 levereras i en 40 pinnars PDIP paket och har många funktioner. De funktioner som vi använde oss av i detta projektet var: 16 bitars timer, timer avbrott, JTAG, USART, I/O pinnar och möjligheten att använda en kristall som extern klocka.

74HC595 Skiftregister

74HC595 är ett 8bitars seriellt till parallellt skiftregister med latch funktion. Detta gör det möjligt att ladda hela registret med data innan utgångarna uppdateras. Utgångarna uppdateras genom att latch pinnen (ST_CP) aktiverad.

MAX232

MAX232 är ett chip som konverterar RS-232 signaler, RS-232 är standarden som används för PCs com-portar, som är +-25V till TTL signaler (0-5 V) som mikrokontrollern klara av.

RGB lysdioder

Lysdioder med 4 pinnar: katod, blå LED, röd LED och grön LED.

MOSFET IRL520N

IRL520N är en N-MOSFET med som går att styra med TTL- nivåer (0-5V) som gatespänning.

Multiplexing

För att minimera antalet kontakter mellan kuben och Mikrokontrollern används tidsmultiplexing. I stället för att tända alla lysdioder i hela kuben samtidigt så tänds ett lager i taget. Om lagren tänds med tillräckligt hög frekvens så uppfattar inte ögat tiden kuben är släckt utan set det som om hela kuben lyser kontinuerligt, som på en TV eller en Datorskärm.

Genom att koppla ihop diodernas anoder så att pelare av dioder bildas minskar antalet kontakter med en faktor 5. Detta leder dock till att hela pelaren tänds samtidigt. Lösningen på detta är att bara jorda ett lager åt gången.

För att jorda ett lager åt gången så är alla dioders katoder i ett lager sammankopplade och kopplade till en transistor. Transistorn fungerar som en switch för att jorda lagret när det ska tändas och för att bryta kontakten när lagret ska vara släckt.

Förberedande beräkningar

För våra beräkningar av strömförbrukning, kostnad mm. se Bilaga *LED, IO, Ström och Priskalkyl*.

Genomförande

Multiplexing

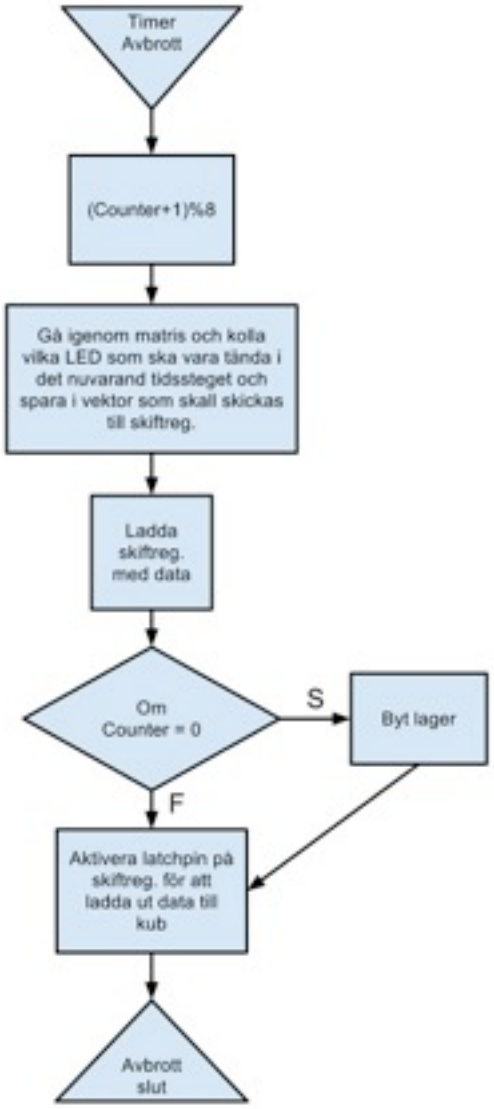
För att minska på antalet I/O som behövs för att styra kuben används tidsmultiplexing av kubens fem lager. På det sättet minskas antalet I/O pinnar från 375 st till 75 I/O för att styra lysdioderna samt 5 st I/O för val av lager, d.v.s. totalt 80st I/O pinnar. Eftersom mikrokontrollern som används inte har 80 st I/O pinnars krävs utbyggnad med hjälp av skiftregister för att utöka antal I/O-pinnar. Skiftregistren som används heter 74HC595 och har den fördelen att de kan kopplas i serie och på så sätt bilda större skiftregister. Detta passar utmärkt till projektet eftersom 10st 74HC595 i serie ger 80st utgångar, samtidigt som de enbart kräver 3st mikrokontroller utgångar för styrning.

Multiplexingen fungerar genom att 74HC595, Skiftregister, laddas med alla värden för ett lager när mikrokontrollern får ett timer avbrott för att uppdatera kuben. Samtidigt väljs det lager som skall tändas genom att aktivera en av de 5 I/O pinnarna som styr transistorerna, som i sin tur är kopplade till de olika lagrens jord pinnar.

Mjukvaru PWM

För att få möjlighet att visa fler än 7 olika färger på RGB lysdioderna används pulsbreddsmodulering (PWM) på lysdioderna för att skapa fler nyanser. Experiment med PWM på en RGB lysdiod visade att 8 ljus nivåer var lagom kompromiss mellan antal färger/nyanser och frekvens på timer-avbrottsrutinen.

Eftersom den mikrokontroller, Atmel ATMega16, som används till LED Kuben bara har 4st hårdvarubaserade PWM-utgångar och kuben kräver 75st för att kunna styra alla lysdioder individuellt, så implementeras PWM i mjukvara. PWM implementerades genom att utnyttja det timer-avbrott som redan används till multiplexing av lager, men istället för att bara byta lager vid varje avbrott så ökades frekvensen så att det går 8 avbrott innan man byter lager. Genom att ha 8 tidssteg på varje lager kan man i ändra pulsbredden till Lysdioden genom att t.ex. låta lysdioden vara tänd dom 2 första tidsstegen och sedan vara släckt i resterande tidssteg och på så sätt få en PWM signal med 25% pulskvot. Hur avbrottsrutinen fungerar kan ses i flödesschema för *avbrottsrutin* nedan.



Flödesschema: *Avbrottsrutin.*

Animationer

Kuben innehåller 11 unika animationer, varav vissa har variationer, vilket get totalt 15 animationer att välja mellan. Hastigheten på alla animationer och färgen på vissa animationer kan ställas via knappsatsen eller serieporten.

FLASHRAND
Hela Kuben tänds i en slumpmässigt vald färg, efter en stund uppdateras kuben med en ny, slumpmässig färg.
CHAOS
Liknande FLASHRAND, men varje enskild diod tänds med en slumpmässig färg.
PLASMA
Kuben simulerar en plasmaeffekt där alla färger cirkulerar runt i olika hastigheter.

CORNER_EXPAND
Ett slumpvist valt hörn tänds med en slumpvis vald färg. Därefter expanderar punkten utåt mot de tre närmaste hörnen.
CORNER_CUBE
CORNER_CUBE börjar på ett liknande sätt som CORNER_EXPAND, men när den expanderar bildas konturen av en kub. När kuben har expanderat maximalt så börja den krympa ihop mot ett ny slumpvis valt hörn. När kuben har krympt till en enda diod väljs en ny färg och animationen börjar om.
CUBE_EXPAND
Dioden mitt i kuben tänds med en slumpvis vald färg. Därefter expanderar den ut tills hela kuben är tänd i den färgen. Dioden i mitten släcks nu och sen släcks de andra dioderna utåt tills kuben åter igen är helt släckt, därefter börjar animationen om.
WORMS
Tre “maskar” gräver sig slumpvist genom kuben. Maskarna har olika färg: röd, grön och blå. Då maskarna hamnar på samma dioder så blandas deras färger.
Denna animation har två varianter. I den första blir maskarna mindre lysande ju längre ifrån huvudet de kommer, medan i den andra lyser hela masken lika starkt.
RAIN
Dioderna högst upp i kuben tänds slumpmässigt och faller därefter ner mot botten på kuben.
Denna animation har två lägen: i det första läget är varje regndroppes färg slumpvis medan i det andra läget har alla droppar samma färg. Färgen i läge två kan bestämmas via knappsatsen eller serieporten.
METEORITES
METEORITES liknar RAIN men då en droppe faller så slutar inte dioden den föll från att lysa, utan lyser bara lite svagare.
Denna animation har två lägen på samma sätt som RAIN.
STAR
Dioderna i kuben tänds slumpmässigt och tonas sedan ner tills de inte lyser mer.
‘Denna animation har två lägen på samma sätt som RAIN.
FIRE
Bilden av en eld simuleras i kuben. Färgen på elden kan ställas in med hjälp av knappsatsen eller via serieporten.

Spel

Kuben innehåller förutom animationerna även två spel.

SNAKE 3D
Detta spel är en en 3D-tolkning av det klassiska spelet SNAKE. Målet är att få ormen att äta de röda prickarna utan att krocka med sig själv.
Styrningen sker med knappsatsen enligt följande:

- I planet så svänger man vänster med vänster knapp och höger med höger. Knappen Upp gör får ormen att röra sig uppåt genom planen och Ner får en att åka ner genom planen.
- När ormen åker mellan planen så svänger den åt det hållet knapparna perkar, dvs upp betyder bort från spelkontrollen osv.

AIR_RAID

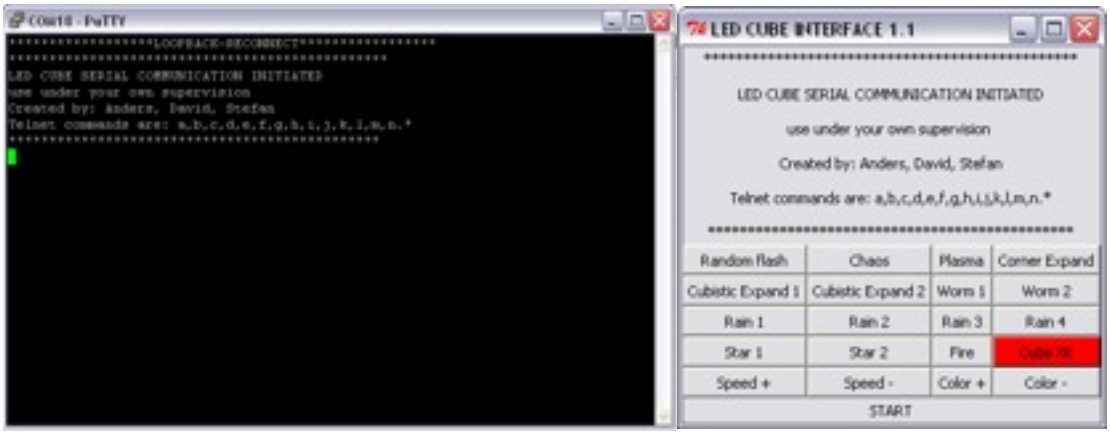
I AIR_RAID gäller det att undvika de fallande röda pixlarna och överleva så länge som möjligt. Spelet går snabbare ju längre tid som går, men om man lyckas fånga upp en fallande blå pixel saktar spelet ner något.

Kommunikation

För att styra ledkubens animationer och spel används dels en knappsats med ett styrkors, dels en seriell UART-port.

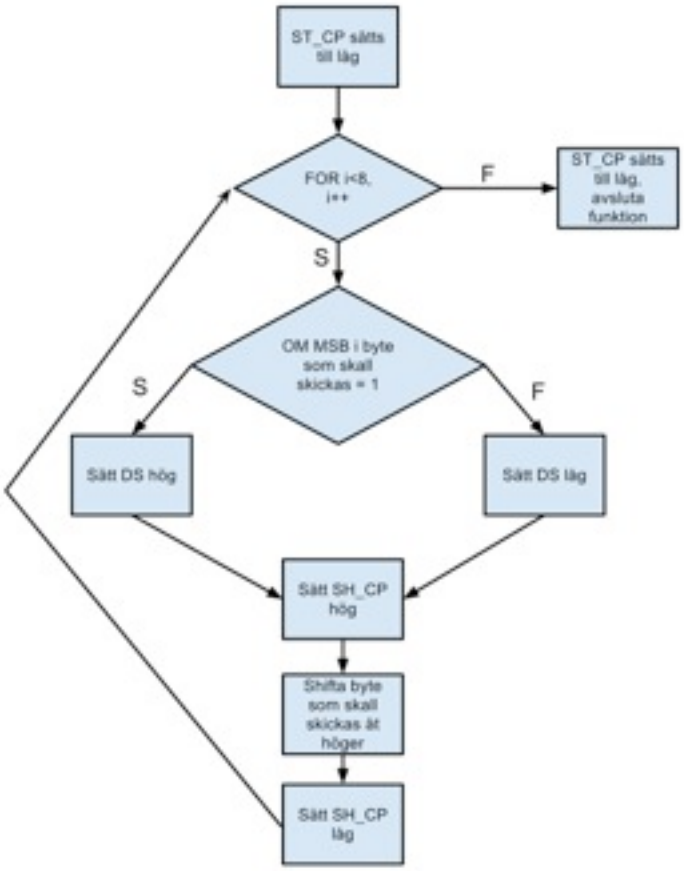
Knappsats
knappsatsen kopplas direkt som insignaler på mikroprocessorn som därefter läser av knapparna genom att kontrollera dess insignaler i main-loopen, så kallad polling.

Serieport
Serieporten ansluts på mikroprocessorn med hjälp av ett max232-chip som omvandlar art signalerna till rätt spänningsnivå för en dator. Till skillnad från knappsatsen använder sig serieporten av interrupts för att styra kuben. Vad gäller kommunikationen här så kan man antingen använda sig av en terminal, ex telnet, eller ett specialskrivet program.
Ansluter man med telnet till kuben svarar den med ett välkomstmeddelande samt en beskrivning av vilka kommandon som kan skickas till den, ex. kommandot för *byt animation* osv. se fig *Telnet* nedan.
När man skickar ett kommando till kuben svarar den genom att utföra åtgärden kopplad till kommandot samt skickar ett svar över serie porten om vilket kommando den utfört.
Specialprogrammet för kommunikation är skrivet i Python och fungerar på samma sätt som *telnet* förutom att kommandona är kopplade till knappar i ett grafiskt interface. se fig *GUI* nedan.



Figur: Från vänster *Telnet*, *GUI*.

Skiftregister interface, 74HC595



Flödesschema: *Interfacerutin för 74HC595.*

För att skicka data till 74HC595 slingan skapades en subrutin som “bit-bangar” de signaler som behövs för att ladda 74HC595 med data. Hur subrutinen fungerar illustreras i flödesschemat för *Interfacerutin för 74HC595.*

Resultat

Ström och energiförbrukning

Eftersom LED kuben är tänkt att användas som konstverk/lampa som är tänd långa stunder åt gången så är en låg energiförbrukning en stor fördel. Enligt våra beräkningar som gjordes innan bygget började så borde kuben dra ca 750mA ström vilket vid 5V motsvarar 3,75W. Dessa beräkningar gjordes medvetet pessimistiska för att kunna ha en viss marginal när vi valde komponenter. Beräkningen gjordes genom att anta att varje lysdiod drar 10mA per färg och sedan beräkna total ström om alla lysdioder i ett lager är tända samtidigt.

När sedan strömförbrukningen mättes på den färdigbyggda kuben visad sig att den endast drar ca 650mA i ström vilket motsvarar 3,25W vid 5V. Minskningen jämfört med beräkningen beror på bl.a. att vi använder PWM och multiplexing och det faktum att de animationer som kuben visar oftast inte tänder alla dioder samtidigt

Utvecklingskostnad

Vi uppskattar att utveckling inklusive montage av själva kuben tagit ca 500 mantimmar, varav ca hälften gått åt till att montera själva hårdvaran. Detta gör att utvecklingskostnaden blir relativt låg i jämförelse med tillverkningskostnaden. Eftersom monteringen är så pass tidskrävande skulle den behöva automatiseras för att möjliggöra lanseringen av en kommersiell produkt. Materialkostnaden per 5x5x5 kub uppgår till 270kr, fördelar vi de övriga kostnaderna över detta får vi kostnaderna i tabellerna: *Kostnadskalkyl* nedan. Vi jämför här dels ett alternativ att sälja en handmonterad kub, där monteringen beräknas kunna fås ner till 8 timmar, dels att sälja kuben som ett byggkitt. Jämförelsen visar även att konkurrensen inte är särskilt hög på marknaden, se diagram *konkurrenssituation*. Dessutom kan man säkert pressa ner inköpspriserna på material och på så vis få ner de materialkostnaderna ytterligare.

MANUELLT MONTERAD	Antal	kr/st	kr/enhet
dm	1	270	270
dL	8	100	800
R&D	450	100	450
Volym	100		
		SJK	1 520,00 kr
vinst pålägg	20%		1 824,00 kr

BYGGKIT	Antal	kr/st	kr/enhet
dm	1	270	270
dL	1	100	100
R&D	450	100	450
Volym	100		
		SJK	820,00 kr
vinst pålägg	20%		984,00 kr

Konkurrenter	Storlek	ex moms	kr ink moms
picprojects.biz	5x5x5	22.79 GBP (ex.led)	500kr (ink. led)
hypnocube	4x4x4	150$	1 230 kr

Tabell: *Kostnadskalkyl*, kuber i jämförelse.

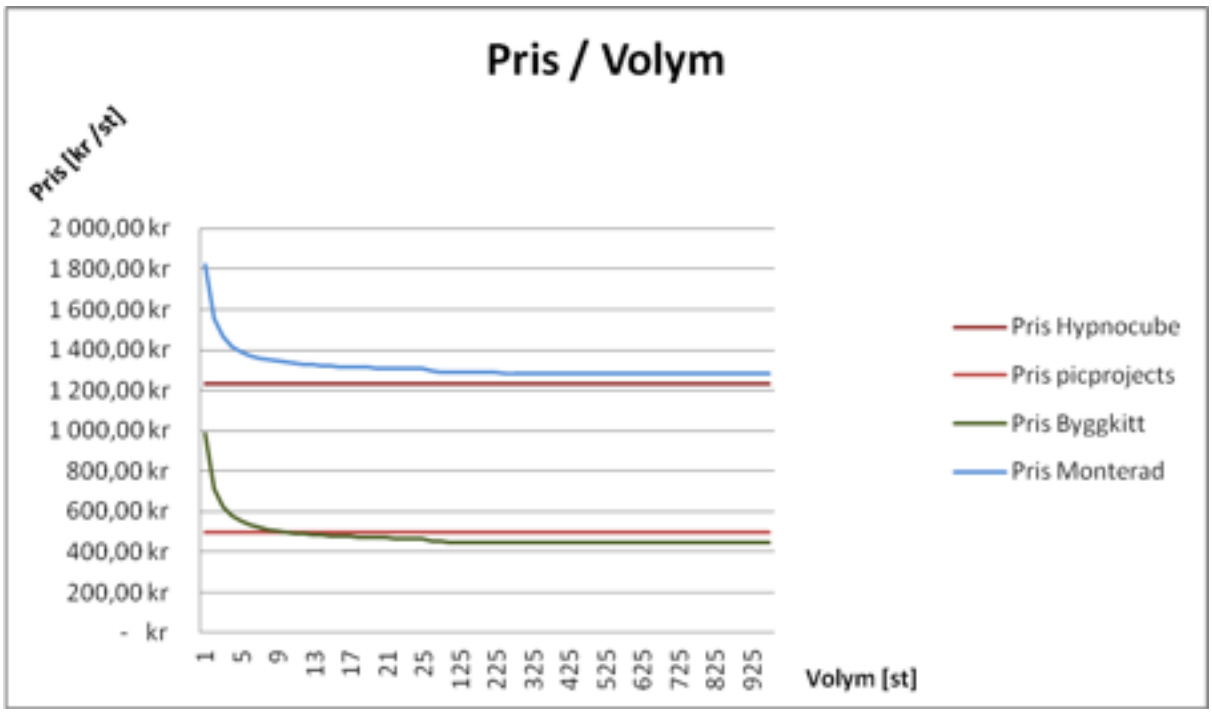


Diagram: *Konkurrenssituation*.

Slutsats

Vi är väldigt nöjda med hur resultatet projektet blev, alla mål som satts i vår kravspecifikation i början av kursen blev uppfyllda. Byggandet av kuben gick väldigt smärtfritt p.g.a. att vi var väldigt noga och testade varje del innan vi kopplade ihop nästa del. Vi hade ett mindre problem med att strömförsörjning blev instabil när alla lysdioder var tända, vilket ledde till att mikrokontrollern hängde sig. Detta löstes genom att öka storleken på de strömbegränsande motstånden och på så sätt sänka strömförbrukningen vilket löste problemet.

Bilaga kod

DRIVER.H

```
#ifndef F_CPU //cpu speed
#define F_CPU 16000000UL //16MHz
#endif

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define CLKPIN (1<<PA0)
#define DATAPIN (1<<PA1)
#define LATCHPIN (1<<PA2)
#define LAYER0 (1<<PA3)
#define LAYER1 (1<<PA4)
#define LAYER2 (1<<PA5)
#define LAYER3 (1<<PA6)
#define LAYER4 (1<<PA7)

#define FCPU 16000000 //16MHz

#define MAX_COLOR 8

typedef unsigned char uint8_t;
typedef unsigned int uint16_t;

void timerSetup(); // Setup of timer registers

void shiftReg_Setup(); // Setup of shiftReg outputs

void shiftData(uint8_t data); // Sends data to 74HC595
shiftregisters

void clear (void); // Clear the cube

void fill (uint8_t r,uint8_t g,uint8_t b); // Fills the
entire cube with a single color

uint8_t put_XYZ (uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b); // Sets the color of a single LED

uint8_t fade_XYZ (uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b); // Fades the color of a single LED

uint8_t tint_XYZ(uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b); // Increase the color of a single
LED

uint8_t single_XYZ (uint8_t x,uint8_t y,uint8_t z,uint8_t
color,uint8_t value); // Tints one color of a single LED

uint8_t getColor(uint8_t x,uint8_t y,uint8_t z,uint8_t
color); //Returns the color of a single LED
```

DRIVER.C

```
/*
 * SoftPWM.c
 *
 * Created: 2012-01-28 20:21:37
 *  Author: Anders & Stefan & David
 */

#include "driver.h"

volatile uint8_t cube[3][5][25]={{{0}}}; //0 red, 1 green, 2
blue


void timerSetup(){

          /** Timer 1 CTC interrupt for display updating **/
          TCCR1A=0; //Normal port operation on OC1A
          TCCR1B |= (1 << WGM12); // Configure timer 1 for
CTC mode
    OCR1A=100; //250 ger intrrupt med 1kHz om F_CPU=16MHz,
http://www.frank-zhao.com/cache/avrtimercalc.php
    TIMSK |= (1 << OCIE1A); // Enable CTC interrupt OC1A
          TCCR1B|=(1<<CS11)|(1<<CS10); // prescale cpu/64

          sei();    //Turn on interrupts
}

void shiftReg_Setup()
{
          DDRA |= (CLKPIN|DATAPIN|LATCHPIN|LAYER0|LAYER1|
LAYER2|LAYER3|LAYER4); //Set pins to output
}


/**************************
* Display update interrupt *
**************************/
ISR(TIMER1_COMPA_vect){
          static uint8_t count=0; //used to PWN
          static uint8_t lager=0;         //used to
multiplex
          uint8_t dataToShift[10]={0,0,0,0,0,0,0,0,0,0};
          uint8_t value=0; //value from matrix

          count=(count+1)%8;


          if (count==0) //change layer
          {
                    lager=(lager+1)%5;
          }

          //loads data from marix in to vector that is sent
to shiftreg.
    value=cube[2][lager][0];  //reg9
          if(value != 0 && (value>=count+1)){
                    dataToShift[0]|=(1<<2);
          }
          value=cube[2][lager][1];
          if(value != 0 && (value>=count+1)){
                    dataToShift[0]|=(1<<1);
          }
          value=cube[2][lager][2];
          if(value != 0 && (value>=count+1)){
                    dataToShift[0]|=(1<<0);
          }
          value=cube[2][lager][3];      //reg8
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<7);
          }
          value=cube[2][lager][4];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<6);
          }
          value=cube[2][lager][5];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<5);
          }
          value=cube[2][lager][6];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<4);
          }
          value=cube[2][lager][7];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=
(1<<3);
          }
          value=cube[2][lager][8];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<2);
          }
          value=cube[2][lager][9];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<1);
          }
          value=cube[2][lager][10];
          if(value != 0 && (value>=count+1)){
                    dataToShift[1]|=(1<<0);
          }
```

```
		value=cube[2][lager][11];	//reg7
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<7);
		}
		value=cube[2][lager][12];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<6);
		}
		value=cube[2][lager][13];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<5);
		}
		value=cube[2][lager][14];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<4);
		}
		value=cube[2][lager][15];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=
(1<<3);
		}
		value=cube[2][lager][16];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<2);
		}
		value=cube[2][lager][17];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<1);
		}
		value=cube[2][lager][18];
		if(value != 0 && (value>=count+1)){
			dataToShift[2]|=(1<<0);
		}
		value=cube[2][lager][19];	//reg6
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<7);
		}
		value=cube[2][lager][20];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<6);
		}
		value=cube[2][lager][21];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<5);
		}
		value=cube[2][lager][22];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<4);
		}
		value=cube[2][lager][23];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<3);
		}
		value=cube[2][lager][24];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<2);
		}
		value=cube[1][lager][0];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<1);
		}
		value=cube[1][lager][1];
		if(value != 0 && (value>=count+1)){
			dataToShift[3]|=(1<<0);
		}
		value=cube[1][lager][2];	//reg5
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<7);
		}
		value=cube[1][lager][3];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<6);
		}
		value=cube[1][lager][4];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<5);
		}
		value=cube[1][lager][5];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<4);
		}
		value=cube[1][lager][6];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=
(1<<3);
		}
		value=cube[1][lager][7];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<2);
		}
		value=cube[1][lager][8];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<1);
		}
		value=cube[1][lager][9];
		if(value != 0 && (value>=count+1)){
			dataToShift[4]|=(1<<0);
		}
		value=cube[1][lager][10];	//reg4
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<7);
		}
		value=cube[1][lager][11];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<6);
		}
		value=cube[1][lager][12];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<5);
		}
		value=cube[1][lager][13];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<4);
		}
		value=cube[1][lager][14];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=
(1<<3);
		}
		value=cube[1][lager][15];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<2);
		}
		value=cube[1][lager][16];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<1);
		}
		value=cube[1][lager][17];
		if(value != 0 && (value>=count+1)){
			dataToShift[5]|=(1<<0);
		}
		value=cube[1][lager][18];	//reg3
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<7);
		}
		value=cube[1][lager][19];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<6);
		}
		value=cube[1][lager][20];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<5);
		}
		value=cube[1][lager][21];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<4);
		}
		value=cube[1][lager][22];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=
(1<<3);
		}
		value=cube[1][lager][23];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<2);
		}
		value=cube[1][lager][24];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<1);
		}
```

```
		value=cube[0][lager][0];
		if(value != 0 && (value>=count+1)){
			dataToShift[6]|=(1<<0);
		}
		value=cube[0][lager][1];    //reg2
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<7);
		}
		value=cube[0][lager][2];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<6);
		}
		value=cube[0][lager][3];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<5);
		}
		value=cube[0][lager][4];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<4);
		}
		value=cube[0][lager][5];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=
(1<<3);
		}
		value=cube[0][lager][6];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<2);
		}
		value=cube[0][lager][7];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<1);
		}
		value=cube[0][lager][8];
		if(value != 0 && (value>=count+1)){
			dataToShift[7]|=(1<<0);
		}
		value=cube[0][lager][9];      //reg1
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<7);
		}
		value=cube[0][lager][10];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<6);
		}
		value=cube[0][lager][11];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<5);
		}
		value=cube[0][lager][12];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<4);
		}
		value=cube[0][lager][13];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=
(1<<3);
		}
		value=cube[0][lager][14];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<2);
		}
		value=cube[0][lager][15];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<1);
		}
		value=cube[0][lager][16];
		if(value != 0 && (value>=count+1)){
			dataToShift[8]|=(1<<0);
		}
		value=cube[0][lager][17];     //reg0
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<7);
		}
		value=cube[0][lager][18];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<6);
		}
		value=cube[0][lager][19];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<5);
		}
		value=cube[0][lager][20];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<4);
		}
		value=cube[0][lager][21];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=
(1<<3);
		}
		value=cube[0][lager][22];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<2);
		}
		value=cube[0][lager][23];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=(1<<1);
		}
		value=cube[0][lager][24];
		if(value != 0 && (value>=count+1)){
			dataToShift[9]|=
(1<<0);
		}


		//LatchPin HIGH
		PORTA|=LATCHPIN;

		/*sending data to shiftreg, one byte at a time*/
		for (uint8_t i=0; i<=9; i++)
		{
			shiftData(dataToShift[i]);
		}

			if (count==0) //layer turn on new layer
		{
			switch (lager)
			{
			case 0:
				PORTA|=LAYER0; //on
				PORTA&=~(LAYER1|LAYER2|LAYER3|
LAYER4); //off
				break;
			case 1:
				PORTA|=LAYER1; //on
				PORTA&=~(LAYER0|LAYER2|LAYER3|
LAYER4); //off
				break;
			case 2:
				PORTA|=LAYER2; //on
				PORTA&=~(LAYER1|LAYER0|LAYER3|
LAYER4); //off
				break;
			case 3:
				PORTA|=LAYER3; //on
				PORTA&=~(LAYER1|LAYER2|LAYER0|
LAYER4); //off
				break;
			case 4:
				PORTA|=LAYER4; //on
				PORTA&=~(LAYER1|LAYER2|LAYER3|
LAYER0); //off
				break;
			}
		}

		//LatchPin LOW, loading values to cube
		PORTA&=~LATCHPIN;
}

void shiftData(uint8_t data){

		for(uint8_t i=0; i<=7; i++){
			if(data & 0b10000000){
				PORTA |= DATAPIN; //set data
to 1
			}else{
				PORTA&=~ DATAPIN; //set data
to 0
```

```
                    }

                    PORTA|= CLKPIN; //clock 1

                    data = data << 1;

                    PORTA&=~CLKPIN; //clock 0
          }
          PORTA&=~ DATAPIN; //set data to 0
}


void clear (void) // Clear the cube
{
    uint8_t ct=0, lv=0;
    for (lv=0;lv<5;lv++)
    {
          for (ct=0;ct<25;ct++)
          {
            cube[0][lv][ct]=0;
            cube[1][lv][ct]=0;
            cube[2][lv][ct]=0;
          }
    }
}

void fill (uint8_t r,uint8_t g,uint8_t b) // Fills the
entire cube with a single color
{
    uint8_t ct=0, lv=0;
    for (lv=0;lv<5;lv++)
    {
          for (ct=0;ct<25;ct++)
          {
            cube[0][lv][ct]=r;
            cube[1][lv][ct]=g;
            cube[2][lv][ct]=b;
          }
    }
}

uint8_t put_XYZ(uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b) // Sets the color of a single LED
{
    uint8_t pixel;

    if (x>4) return 1;
    if (y>4) return 2;
    if (z>4) return 3;

    pixel=(x)+(z*5);

    cube[0][y][pixel]=r;
    cube[1][y][pixel]=g;
    cube[2][y][pixel]=b;
    return 0;
}

uint8_t fade_XYZ(uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b) // Fades the color of a single LED
{
    uint8_t pixel;

    if (x>4) return 1;
    if (y>4) return 2;
    if (z>4) return 3;

    pixel=(x)+(z*5);

          if(cube[0][y][pixel]<=r){
                    cube[0][y][pixel]=0;
          }else{
                    cube[0][y][pixel]-=r;
          }
          if(cube[1][y][pixel]<=g){
                    cube[1][y][pixel]=0;
          }else{
                    cube[1][y][pixel]-=g;
          }
          if(cube[2][y][pixel]<=b){
                    cube[2][y][pixel]=0;
          }else{
                    cube[2][y][pixel]-=b;
          }
    return 0;
}

uint8_t tint_XYZ(uint8_t x,uint8_t y,uint8_t z,uint8_t
r,uint8_t g,uint8_t b) // Increase the color of a single LED
{
    uint8_t pixel;

    if (x>4) return 1;
    if (y>4) return 2;
    if (z>4) return 3;

    pixel=(x)+(z*5);

          cube[0][y][pixel]+=r;
          cube[1][y][pixel]+=g;
          cube[2][y][pixel]+=b;

          if(cube[0][y][pixel]>7){
                    cube[0][y][pixel]=7;
          }
          if(cube[1][y][pixel]>7){
                    cube[1][y][pixel]=7;
          }
          if(cube[2][y][pixel]>7){
                    cube[2][y][pixel]=7;
          }
    return 0;
}

uint8_t single_XYZ (uint8_t x,uint8_t y,uint8_t z,uint8_t
color,uint8_t value) // Tints one color of a single LED
{
    uint8_t pixel;

    if (x>4) return 1;
    if (y>4) return 2;
    if (z>4) return 3;
          if (color>2) return 4;

    pixel=(x)+(z*5);

    cube[color][y][pixel]+=value;
    if(cube[color][y][pixel] > 7)
    {
                    cube[color][y][pixel] = 7;
    }
    return 0;
}

uint8_t getColor(uint8_t x,uint8_t y,uint8_t z,uint8_t
color) //Returns the color of a single LED
{
          uint8_t pixel;

          if (x>4) return 0;
    if (y>4) return 0;
    if (z>4) return 0;
          if (color>2) return 0;

    pixel=(x)+(z*5);

    return cube[color][y][pixel];
}
```

PATTERNS.H

```
#include <math.h>
#include <avr/pgmspace.h>
#include <stdlib.h>


void blackout(uint16_t time);
```

```
void worm (uint8_t solid, uint8_t time);

void flash_rand (uint8_t time);

void rain (uint8_t mode, uint8_t afterglow, uint8_t c,
uint8_t time);

void star (uint8_t mode, uint8_t c, uint8_t time);

void corner_expand (uint8_t time);

void corner_cube (uint8_t time);

void plasma (uint8_t time);

void chaos(uint8_t time);

void cube_expand(uint8_t time);

void fire(uint8_t c, uint8_t time);

void display_colors (uint16_t len);

void airRaidReset(void);

void wormsReset(void);

uint8_t airRaid(uint8_t x, uint8_t z);

uint8_t Snake(uint8_t dir);

void gameOver(void);
```

PATTERNS.C

```
#include "patterns.h"
#include "test_patterns.h"
#include "driver.h"

const uint8_t PROGMEM redData[]=
{7,7,7,7,7,7,7,6,5,4,3,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
,2,3,4,5,6,7,7,7,7,7,7,7,7};
const uint8_t PROGMEM greenData[]=
{0,1,2,3,4,5,6,7,7,7,7,7,7,7,7,7,7,7,7,7,6,5,4,3,2,1,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,7};
const uint8_t PROGMEM blueData[]=
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,5,6,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,6,5,4,3,2,1,7};
const uint8_t PROGMEM mysin[]=
{0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2
,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4
,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5
,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,5,5,5,5,5,5
,5,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

static uint8_t col[5][5][2]={{{0}}};
static uint8_t worms[3][4][3]={{{0}}};

void blackout(uint16_t time)
{
    clear();
    my_delay_ms(time,0,0,0);
}

void worm (uint8_t solid, uint8_t time)
{
    uint8_t test[3]={0}, direction=0, p_m=0, ct=0,
collision, colorct=0;
          clear();
    for (colorct=0;colorct<3;colorct++) // for RGB worms
    {
        collision=1;
        while (collision)
        {
                collision=0;
                test[0]=worms[colorct][0][0];
                test[1]=worms[colorct][0][1];
                test[2]=worms[colorct][0][2];
                direction=rand() % 3; //choose X,Y, or Z
                p_m=rand() % 2; //choose +/- on the axis
                if (p_m) test[direction]++;
                else test[direction]--;
                if (test[direction]>4) collision=1; //are we
outside the cube? if so we need a new direction
                else
                //does the new point interset with any part
of the same color worm?
                if (((test[0]==worms[colorct][1][0])&&(test
[1]==worms[colorct][1][1])&&(test[2]==worms[colorct][1]
[2]))||
                    ((test[0]==worms[colorct][2][0])&&(test
[1]==worms[colorct][2][1])&&(test[2]==worms[colorct][2]
[2]))||
                    ((test[0]==worms[colorct][3][0])&&(test
[1]==worms[colorct][3][1])&&(test[2]==worms[colorct][3]
[2]))) collision=1;
                    }

            for (ct=3;ct>0;ct--)
            {
                //update the tail
                worms[colorct][ct][0]=worms[colorct][ct-1]
[0];
                worms[colorct][ct][1]=worms[colorct][ct-1]
[1];
                worms[colorct][ct][2]=worms[colorct][ct-1]
[2];
            }

            //set the leading point
            worms[colorct][0][0]=test[0];
            worms[colorct][0][1]=test[1];
            worms[colorct][0][2]=test[2];


          //display the worm
                    if(solid == 0)
                    {
                    single_XYZ(worms[colorct][3][0],worms
[colorct][3][1],worms[colorct][3][2],colorct,1);
                    single_XYZ(worms[colorct][2][0],worms
[colorct][2][1],worms[colorct][2][2],colorct,3);
                    single_XYZ(worms[colorct][1][0],worms
[colorct][1][1],worms[colorct][1][2],colorct,5);
                    single_XYZ(worms[colorct][0][0],worms
[colorct][0][1],worms[colorct][0][2],colorct,7);
                    }else if(solid == 1)
                    {
                                        single_XYZ(worms
[colorct][3][0],worms[colorct][3][1],worms[colorct][3]
[2],colorct,7);
                    single_XYZ(worms[colorct][2][0],worms
[colorct][2][1],worms[colorct][2][2],colorct,7);
                    single_XYZ(worms[colorct][1][0],worms
[colorct][1][1],worms[colorct][1][2],colorct,7);
                    single_XYZ(worms[colorct][0][0],worms
[colorct][0][1],worms[colorct][0][2],colorct,7);
                    }
        }
        if(my_delay_ms(10*(2+time),1,1,0)) //slow them down
                    {
                              wormsReset();
                    }
}

void flash_rand (uint8_t time)
{
    uint16_t hue=0;
    hue = rand() % 43;
    fill(pgm_read_byte(&redData[hue]),pgm_read_byte
(&greenData[hue]),pgm_read_byte(&blueData[hue]));
```

```
    my_delay_ms(65*(3+time),1,1,0);
}


void rain (uint8_t mode, uint8_t afterglow, uint8_t c,
uint8_t time)
{
    uint8_t set_x=0,set_y=0,set_z=0,r=0,g=0,b=0,hue=0;
    for (set_x=0;set_x<5;set_x++)
    {
                    for (set_z=0;set_z<5;set_z++)
        {
                                for (set_y=0;set_y<4;set_y++)
                                {
                                r = getColor(set_x,set_y
+1,set_z,0);
                                g = getColor(set_x,set_y
+1,set_z,1);
                                b = getColor(set_x,set_y
+1,set_z,2);
                put_XYZ(set_x,set_y,set_z,r,g,b);
                                }
                                if (rand() % (7*
(1+5*afterglow)) == 0)
                                {
                                if(mode == 0)
                                {
                                                hue = rand
() % 43;
                                }else
                                {
                                                hue = c;
                                }
                                        put_XYZ(set_x,
4,set_z,
                    pgm_read_byte(&redData[hue]),
                    pgm_read_byte(&greenData[hue]),
                    pgm_read_byte(&blueData[hue]));
                                }else{
                                if(afterglow == 0)
                                {
                                        put_XYZ(set_x,
4,set_z,0,0,0);
                                }else
                                {
                                                fade_XYZ
(set_x,4,set_z,1,1,1);
                                }
                                }
        }
    }
    my_delay_ms(10*(1+time),1,1,0);
}

void star (uint8_t mode, uint8_t c, uint8_t time)
{
    uint8_t set_x=0,set_y=0,set_z=0,hue=0;
    for (set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                                for (set_y=0;set_y<5;set_y++)
                                {
                                fade_XYZ(set_x,set_y,set_z,
1,1,1);
                                if (rand() % 130 == 0)
                                {
                                        if(mode == 0)
                                        {
                                                hue = rand
() % 43;
                                        }else
                                        {
                                                hue = c;
                                        }
                                        put_XYZ
(set_x,set_y,set_z,
                                pgm_read_byte(&redData[hue]),
                                pgm_read_byte(&greenData
[hue]),
                                pgm_read_byte(&blueData
[hue]));
                                }
                                }
                    }
    }
    my_delay_ms(13*(1+time),1,1,0);
}

void corner_expand (uint8_t time)
{
    uint8_t corner[3]={0}, color[3]={0}, ct=0, hue=0;
    hue = rand() % 43;
    color[0]=pgm_read_byte(&redData[hue]);
    color[1]=pgm_read_byte(&greenData[hue]);
    color[2]=pgm_read_byte(&blueData[hue]);

    for (ct=0;ct<3;ct++)
    {
        corner[ct]=(rand() % 2)*4;
    }

          clear();
    for(ct = 0; ct<5;ct++)
    {
        if (corner[0]+ct<=4)
        put_XYZ(corner[0]+ct,corner[1],corner[2],color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0]-ct,corner[1],corner[2],color
[0],color[1],color[2]);

        if (corner[1]+ct<=4)
        put_XYZ(corner[0],corner[1]+ct,corner[2],color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0],corner[1]-ct,corner[2],color
[0],color[1],color[2]);

        if (corner[2]+ct<=4)
        put_XYZ(corner[0],corner[1],corner[2]+ct,color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0],corner[1],corner[2]-ct,color
[0],color[1],color[2]);

        if(my_delay_ms(5*(1+time),1,1,0))
                    {
                                return;
                    }
    }
        my_delay_ms(17*(1+time),1,1,0);
}

void corner_cube (uint8_t time)
{
    uint8_t color[3]={0}, ct=0, n=0, hue=0, x1 = 0, x2=0, y1
= 0, y2=0, z1 = 0, z2=0;
    static uint8_t corner[2][3]={{0}};
          hue = rand() % 43;
    color[0]=pgm_read_byte(&redData[hue]);
    color[1]=pgm_read_byte(&greenData[hue]);
    color[2]=pgm_read_byte(&blueData[hue]);

    for (ct=0;ct<3;ct++)
    {
                    corner[0][ct]=corner[1][ct];
        corner[1][ct]=(rand() % 2)*4;
    }

          for(n = 0; n<2;n++)
          {
          for(ct = 0; ct<5;ct++)
          {
                                if(corner[n][0] == 0)
                                {
                                        x1 = 0;
                                        if(n==0)
                                                x2 = ct;
                                        else
```

```
                                        x2 = 4-ct;
                                }else{
                                        x2 = 4;
                                        if(n==0)
                                                x1 = 4-ct;
                                        else
                                                x1 = ct;
                                }
                                if(corner[n][1] == 0)
                                {
                                        y1 = 0;
                                        if(n==0)
                                                y2 = ct;
                                        else
                                                y2 = 4-ct;
                                }else{
                                        y2 = 4;
                                        if(n==0)
                                                y1 = 4-ct;
                                        else
                                                y1 = ct;
                                }
                                if(corner[n][2] == 0)
                                {
                                        z1 = 0;
                                        if(n==0)
                                                z2 = ct;
                                        else
                                                z2 = 4-ct;
                                }else{
                                        z2 = 4;
                                        if(n==0)
                                                z1 = 4-ct;
                                        else
                                                z1 = ct;
                                }
                                clear();
                                for (uint8_t x=x1;x<=x2;x++)
                        {
                        for (uint8_t y=y1;y<=y2;y++)
                        {
                                for (uint8_t z=z1;z<=z2;z++)
                                {

if((x == x1 || x == x2) && (y == y1 || y == y2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

if((x == x1 || x == x2) && (z == z1 || z == z2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

if((y == y1 || y == y2) && (z == z1 || z == z2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

                                        }
                                }
                        }
                if(my_delay_ms(17*(1+time),1,1,0))
                                {
                                        for (ct=0;ct<3;ct++)
                                    {
                                        corner[1][ct]=(rand()
% 2)*4;
                                    }
                                        return;
                                }
                }
                if(my_delay_ms(17*(1+time),1,1,0))
                {
                          for (ct=0;ct<3;ct++)
                    {
                        corner[1][ct]=(rand() % 2)*4;
                    }
                }
}

void plasma (uint8_t time)
{
    uint8_t lx=0,ly=0,lz=0,vr=0,vg=0,vb=0;
          int r=0,g=0,b=0;
    static uint16_t radian_r=0,radian_g=0,radian_b=0;
    radian_r+=2;
    if (radian_r>=314)
          {
        radian_r-=314;
          }
    radian_g+=6;
    if (radian_g>=300)
          {
        radian_g-=300;
          }
    radian_b+=7;
    if (radian_b>=300)
    {
              radian_b-=300;
          }
    for (lx=0;lx<5;lx++)
    {
          for (ly=0;ly<5;ly++)
        {
          for (lz=0;lz<5;lz++)
            {
                                        r = (radian_r+20*
(lx-ly+lz));
                                        if (r>=314)
                                        {
                                    r-=314;
                                        }else if(r<0){
                                                  r+=314;
                                        }
                                        g = (radian_g+20*
(ly-lz+lx));
                                        if (g>=300)
                                        {
                                    g-=300;
                                        }else if(g<0){
                                                  g+=300;
                                        }
                                        b = (radian_b+20*
(lz-lx+ly));
                                        if (b>=300)
                                        {
                                    b-=300;
                                        }else if(b<0){
                                                  b+=300;
                                        }
                                        vr = 7-pgm_read_byte
(&mysin[r]);
                                        vg = 7-pgm_read_byte
(&mysin[g]);
                                        vb = 7-pgm_read_byte
(&mysin[b]);
                                        put_XYZ
(lx,ly,lz,vr,vg,vb);
                                  }
                        }
    }
    my_delay_ms(2*(3+time),1,1,0);
}

void chaos(uint8_t time)
{
          uint8_t hue=0;
          for (uint8_t x=0;x<5;x++)
```

```
    {
        for (uint8_t y=0;y<5;y++)
        {
            for (uint8_t z=0;z<5;z++)
            {
                                        hue = rand() % 43;
                                        put_XYZ(x,y,z,

          pgm_read_byte(&redData[hue]),

          pgm_read_byte(&greenData[hue]),

          pgm_read_byte(&blueData[hue]));
                                }
                    }
          }
          my_delay_ms(26*(1+time),1,1,0);
}

void cube_expand(uint8_t time)
{
          uint8_t hue=0;
          hue = rand() % 43;
          for(uint8_t n=0;n<2;n++)
          {
                    for(uint8_t diff=0;diff<3;diff++)
                    {
                              for (uint8_t x=2-
diff;x<=2+diff;x++)
                    {
                    for (uint8_t y=2-diff;y<=2+diff;y++)
                    {
                              for (uint8_t z=2-
diff;z<=2+diff;z++)
                              {

if(n == 0)

{



          put_XYZ(x,y,z,

                    pgm_read_byte(&redData[hue]),

                    pgm_read_byte(&greenData[hue]),

                    pgm_read_byte(&blueData[hue]));

}else{

          put_XYZ(x,y,z,0,0,0);

}
                                                  }
                                        }
                              }
                              if(my_delay_ms(25*(1+time),
1,1,0))
                              {
                                        return;
                              }
                    }
                    if(my_delay_ms(50*(1+time),1,1,0))
                    {
                              return;
                    }
          }
}

void fire(uint8_t c, uint8_t time)
{
    static uint8_t col2[5][5]={{0}};
    uint8_t set_x=0,set_z=0,rnd=0,r=0,g=0,b=0,max=0;
          r=pgm_read_byte(&redData[c]);
    g=pgm_read_byte(&greenData[c]);
    b=pgm_read_byte(&blueData[c]);
          for(set_x=0;set_x<5;set_x++)
    {
          for (set_z=0;set_z<5;set_z++)
          {
                              rnd = rand()%7;
                              if(rnd == 0)
                              {
                                        if((set_x == 0 ||
set_x == 4) || (set_z == 0 || set_z == 4))
                                        {
                                                  max = 2;
                                        } else if((set_x ==
1 || set_x == 3) || (set_z == 1 || set_z == 3))
                                        {
                                                  max = 4;
                                        } else
                                        {
                                                  max = 5;
                                        }
                                        if(col2[set_x]
[set_z]!=max)
                                        {
                                                  col2
[set_x][set_z]++;
                                        }
                              }else if(rnd == 6)
                              {
                                        if(col2[set_x]
[set_z]!=0)
                                        {
                                                  col2
[set_x][set_z]--;
                                        }
                              }
               }
           }
           col2[0][0]=0;
           col2[0][4]=0;
           col2[4][0]=0;
           col2[4][4]=0;
    for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                              for(uint8_t y=0;y<5;y++)
                              {
                              if(col2[set_x][set_z] > y)
                                        {
                                                  put_XYZ
(set_x,y,set_z,r,g,b);
                                                  if((set_x
== 0 || set_x == 4) || (set_z == 0 || set_z == 4))
                                                  {

fade_XYZ(set_x,y,set_z,6,6,6);
                                                  } else if
((set_x == 1 || set_x == 3) || (set_z == 1 || set_z == 3))
                                                  {

fade_XYZ(set_x,y,set_z,3,3,3);
                                                  } else if
(y > 2)
                                                  {

fade_XYZ(set_x,y,set_z,y-1,y-1,y-1);
                                                  }

                                        }else
                                        {
                                                  fade_XYZ
(set_x,y,set_z,1,1,1);
                                        }
                              }
                    }
    }
    if(my_delay_ms(3*(1+time),1,1,0))
          {
                    for(set_x=0;set_x<5;set_x++)
               {
                   for (set_z=0;set_z<5;set_z++)
                   {
```

```
                                        col2[set_x][set_z]
=0;
                            }
                    }
          }
}

void display_colors (uint16_t len)
{
    uint8_t lx=0,ly=0,lz=0,r=0,g=0,b=0;
    clear();
    for (lx=0;lx<5;lx++)
    {
        for (ly=0;ly<5;ly++)
        {
            for (lz=0;lz<5;lz++)
            {
                                        if(lx == 0)
                                        {
                                                r = 0;
                                        }else
                                        {
                                                r =
2*lx-1;
                                        }
                                        if(ly == 0)
                                        {
                                                g = 0;
                                        }else
                                        {
                                                g =
2*ly-1;
                                        }
                                        if(lz == 0)
                                        {
                                                b = 0;
                                        }else
                                        {
                                                b =
2*lz-1;
                                        }
                put_XYZ(lx,ly,lz,r,g,b);
            }
        }
    }
    my_delay_ms(len,0,0,0);
}

void airRaidReset(void)
{
          uint8_t set_x=0,set_z=0;
          for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                            col[set_x][set_z][0]=5;
             }
          }
}

void wormsReset(void)
{
          for(uint8_t m = 0; m < 3;m++)
          {
                    for(uint8_t n = 0; n < 4;n++)
                    {
                              worms[m][n][0]=2;
                              worms[m][n][1]=2;
                              worms[m][n][2]=2;
                    }
          }
}

uint8_t airRaid(uint8_t x, uint8_t z)
{
    static uint8_t cnt=0,s=0,t=0;
    uint8_t set_x=0,set_z=0;

    if(col[x][z][0]==4)
    {
                    if(col[x][z][1]==0)
                    {
                              clear();
                              for(set_x=0;set_x<5;set_x++)
                    {
                    for (set_z=0;set_z<5;set_z++)
                    {
                                                  if(col
[set_x][set_z][0] < 5)
                                                  {
                                                  single_XYZ
(set_x,4-col[set_x][set_z][0],set_z,2*col[set_x][set_z][1],
7);
                                                  }
                                        }
                    }
                              put_XYZ (x,0,z,7,7,0);
                              gameOver();
                              s=0;
                              t=0;
                              airRaidReset();
                              return 0;
                    }else
                    {
                              if(s < 5)
                              {
                                        s = 0;
                              }else
                              {
                                        s -= 5;
                              }
                    }
    }

    if(cnt != 4)
    {
                    cnt++;
    }else
    {
                    for(set_x=0;set_x<5;set_x++)
          {
          for (set_z=0;set_z<5;set_z++)
          {
                                        col[set_x][set_z]
[0]++;
                    }
                    }
                    if (col[set_x = rand() % 5][set_z = rand
() % 5][0]>4)
          {
          col[set_x][set_z][0]=0;
                              if(rand()%100 <= 3)
                              {
                                        col[set_x][set_z][1]
=1;
                              }else
                              {
                                        col[set_x][set_z][1]
=0;
                              }
          }
                    cnt=0;
    }

          if(t != 13)
    {
                    t++;
    }else
    {
                    if(s<30)
                    {
                              s++;
                    }
                    t=0;
    }

          clear();
    for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
```

```
        {
                                if(col[set_x][set_z][0] < 5){
                                single_XYZ (set_x,4-col[set_x]
[set_z][0],set_z,2*col[set_x][set_z][1],7);
                                }
                        }
    }
    single_XYZ (x,0,z,1,7);
    if(my_delay_ms(50-s,0,1,0))
          {
                    airRaidReset();
                    s=0;
                    t=0;
                    return 0;
          }
    return 1;
}

uint8_t Snake(uint8_t dir)
{
    static uint8_t snake[24][3]={{1,2,2}}, apple[3]={1,2,2},
len=0, crt=0;
          uint8_t n=0,x=0,y=0,z=0;

          void reset(void)
          {
                    len = 0;
                    crt = 10;
                    snake[0][0]=1;
                    snake[0][1]=2;
                    snake[0][2]=2;
                    apple[0]=1;
                    apple[1]=2;
                    apple[2]=2;
          }

          uint8_t colission()
          {
                    if(getColor(snake[0][0],snake[0]
[1],snake[0][2],1)==0)
                    {
                              return 0;
                    }
                    return 1;
          }

          uint8_t gotApple()
          {
                    if(snake[0][0]==apple[0] && snake[0][1]
==apple[1] && snake[0][2]==apple[2])
                    {
                              return 1;
                    }
                    return 0;
          }

          if(gotApple() == 1)
          {
                    while(getColor(x = rand() % 5,y = rand()
% 5,z = rand() % 5,2)!=0)
                    {
                    }
                    apple[0]=x;
                    apple[1]=y;
                    apple[2]=z;
                    if(len != 23)
                    {
                    len++;
                    }
          }

    if(crt != 50)
    {
                    crt++;
    }else
    {
                    if(len != 23)
                    {
                    len++;
                    }
                    crt = 0;
    }

    for(n=len;n>0;n--)
    {
                    snake[n][0]=snake[n-1][0];
                    snake[n][1]=snake[n-1][1];
                    snake[n][2]=snake[n-1][2];
    }

    switch (dir){
          case 0:
                              if(snake[0][0]==4)
                              {
                              snake[0][0]=0;
                              }else
                              {
                              snake[0][0]++;
                              }
                    break;
          case 1:
                              if(snake[0][2]==4)
                              {
                              snake[0][2]=0;
                              }else
                              {
                              snake[0][2]++;
                              }
                    break;
          case 2:
                              if(snake[0][0]==0)
                              {
                                  snake[0][0]=4;
                              }else
                              {
                                        snake[0][0]--;
                              }
                    break;
          case 3:
                              if(snake[0][2]==0)
                              {
                              snake[0][2]=4;
                              }else
                              {
                                  snake[0][2]--;
                              }
                    break;
          case 4:
                              if(snake[0][1]==4)
                              {
                                        snake[0][1]=0;
                              }else
                              {
                              snake[0][1]++;
                              }
                    break;
          case 5:
                              if(snake[0][1]==0)
                              {
                              snake[0][1]=4;
                              }else
                              {
                              snake[0][1]--;
                              }
                    break;
    }

    if(colission() == 1){
                    gameOver();
                    reset();
                    return 0;
    }

          clear();

          put_XYZ(apple[0],apple[1],apple[2],7,0,0);
          put_XYZ(snake[0][0],snake[0][1],snake[0][2],
0,7,0);

          for(n=1;n<=len;n++)
```

```
        {

                put_XYZ(snake[n][0],snake[n][1],snake[n]
[2],7,7,0);

        }

        if(my_delay_ms(250,0,1,1))
        {
                reset();
                return 0;
        }
        return 1;

}

void gameOver(void)
{
        my_delay_ms(500,0,0,0);
        blackout(100);
        display_colors(500);
        blackout(100);

}
```

TEST_PATTERNS.H

```
#include "patterns.h"
#include "test_patterns.h"
#include "driver.h"

const uint8_t PROGMEM redData[]=
{7,7,7,7,7,7,7,6,5,4,3,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
,2,3,4,5,6,7,7,7,7,7,7,7,7};
const uint8_t PROGMEM greenData[]=
{0,1,2,3,4,5,6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,6,5,4,3,2,1,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,7};
const uint8_t PROGMEM blueData[]=
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,5,6,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,6,5,4,3,2,1,7};
const uint8_t PROGMEM mysin[]=
{0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2
,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4
,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5
,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,5,5,5,5,5,5
,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

static uint8_t col[5][5][2]={{{0}}};
static uint8_t worms[3][4][3]={{{0}}};

void blackout(uint16_t time)
{
    clear();
    my_delay_ms(time,0,0,0);
}

void worm (uint8_t solid, uint8_t time)
{
    uint8_t test[3]={0}, direction=0, p_m=0, ct=0,
collision, colorct=0;
          clear();
    for (colorct=0;colorct<3;colorct++) // for RGB worms
    {
        collision=1;
        while (collision)
        {
                collision=0;
                test[0]=worms[colorct][0][0];
                test[1]=worms[colorct][0][1];
                test[2]=worms[colorct][0][2];
                direction=rand() % 3; //choose X,Y, or Z
                p_m=rand() % 2; //choose +/- on the axis
                if (p_m) test[direction]++;
                else test[direction]--;
                if (test[direction]>4) collision=1; //are we
outside the cube? if so we need a new direction
                else
                //does the new point interset with any part
of the same color worm?
                if (((test[0]==worms[colorct][1][0])&&(test
[1]==worms[colorct][1][1])&&(test[2]==worms[colorct][1]
[2]))||
                    ((test[0]==worms[colorct][2][0])&&(test
[1]==worms[colorct][2][1])&&(test[2]==worms[colorct][2]
[2]))||
                    ((test[0]==worms[colorct][3][0])&&(test
[1]==worms[colorct][3][1])&&(test[2]==worms[colorct][3]
[2]))) collision=1;
                }

            for (ct=3;ct>0;ct--)
            {
                //update the tail
                worms[colorct][ct][0]=worms[colorct][ct-1]
[0];
                worms[colorct][ct][1]=worms[colorct][ct-1]
[1];
                worms[colorct][ct][2]=worms[colorct][ct-1]
[2];
            }

            //set the leading point
            worms[colorct][0][0]=test[0];
            worms[colorct][0][1]=test[1];
            worms[colorct][0][2]=test[2];


          //display the worm
                    if(solid == 0)
                    {
                    single_XYZ(worms[colorct][3][0],worms
[colorct][3][1],worms[colorct][3][2],colorct,1);
                    single_XYZ(worms[colorct][2][0],worms
[colorct][2][1],worms[colorct][2][2],colorct,3);
                    single_XYZ(worms[colorct][1][0],worms
[colorct][1][1],worms[colorct][1][2],colorct,5);
                    single_XYZ(worms[colorct][0][0],worms
[colorct][0][1],worms[colorct][0][2],colorct,7);
                    }else if(solid == 1)
                    {
                                        single_XYZ(worms
[colorct][3][0],worms[colorct][3][1],worms[colorct][3]
[2],colorct,7);
                    single_XYZ(worms[colorct][2][0],worms
[colorct][2][1],worms[colorct][2][2],colorct,7);
                    single_XYZ(worms[colorct][1][0],worms
[colorct][1][1],worms[colorct][1][2],colorct,7);
                    single_XYZ(worms[colorct][0][0],worms
[colorct][0][1],worms[colorct][0][2],colorct,7);
                    }
        }
        if(my_delay_ms(10*(2+time),1,1,0)) //slow them down
                    {
                            wormsReset();
                    }
}

void flash_rand (uint8_t time)
{
    uint16_t hue=0;
    hue = rand() % 43;
    fill(pgm_read_byte(&redData[hue]),pgm_read_byte
(&greenData[hue]),pgm_read_byte(&blueData[hue]));
    my_delay_ms(65*(3+time),1,1,0);
}


void rain (uint8_t mode, uint8_t afterglow, uint8_t c,
uint8_t time)
{
    uint8_t set_x=0,set_y=0,set_z=0,r=0,g=0,b=0,hue=0;
```

```
    for (set_x=0;set_x<5;set_x++)
    {
                    for (set_z=0;set_z<5;set_z++)
        {
                                for (set_y=0;set_y<4;set_y++)
                                {
                                r = getColor(set_x,set_y
+1,set_z,0);
                                g = getColor(set_x,set_y
+1,set_z,1);
                                b = getColor(set_x,set_y
+1,set_z,2);
                put_XYZ(set_x,set_y,set_z,r,g,b);
                                }
                                if (rand() % (7*
(1+5*afterglow)) == 0)
                                {
                                if(mode == 0)
                                {
                                                hue = rand
() % 43;
                                }else
                                {
                                                hue = c;
                                }
                                        put_XYZ(set_x,
4,set_z,
                    pgm_read_byte(&redData[hue]),
                    pgm_read_byte(&greenData[hue]),
                    pgm_read_byte(&blueData[hue]));
                                }else{
                                if(afterglow == 0)
                                {
                                        put_XYZ(set_x,
4,set_z,0,0,0);
                                }else
                                {
                                                fade_XYZ
(set_x,4,set_z,1,1,1);
                                }
                                }
        }
    }
    my_delay_ms(10*(1+time),1,1,0);
}

void star (uint8_t mode, uint8_t c, uint8_t time)
{
    uint8_t set_x=0,set_y=0,set_z=0,hue=0;
    for (set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                                for (set_y=0;set_y<5;set_y++)
                                {
                                fade_XYZ(set_x,set_y,set_z,
1,1,1);
                                if (rand() % 130 == 0)
                                {
                                        if(mode == 0)
                                        {
                                                hue = rand
() % 43;
                                        }else
                                        {
                                                hue = c;
                                        }
                                        put_XYZ
(set_x,set_y,set_z,
                                pgm_read_byte(&redData[hue]),
                                pgm_read_byte(&greenData
[hue]),
                                pgm_read_byte(&blueData
[hue]));
                                }
                                }
                    }
    }
    my_delay_ms(13*(1+time),1,1,0);
}
```

```
void corner_expand (uint8_t time)
{
    uint8_t corner[3]={0}, color[3]={0}, ct=0, hue=0;
    hue = rand() % 43;
    color[0]=pgm_read_byte(&redData[hue]);
    color[1]=pgm_read_byte(&greenData[hue]);
    color[2]=pgm_read_byte(&blueData[hue]);

    for (ct=0;ct<3;ct++)
    {
        corner[ct]=(rand() % 2)*4;
    }

          clear();
    for(ct = 0; ct<5;ct++)
    {
        if (corner[0]+ct<=4)
        put_XYZ(corner[0]+ct,corner[1],corner[2],color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0]-ct,corner[1],corner[2],color
[0],color[1],color[2]);

        if (corner[1]+ct<=4)
        put_XYZ(corner[0],corner[1]+ct,corner[2],color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0],corner[1]-ct,corner[2],color
[0],color[1],color[2]);

        if (corner[2]+ct<=4)
        put_XYZ(corner[0],corner[1],corner[2]+ct,color
[0],color[1],color[2]);
        else
        put_XYZ(corner[0],corner[1],corner[2]-ct,color
[0],color[1],color[2]);

        if(my_delay_ms(5*(1+time),1,1,0))
                    {
                            return;
                    }
    }
        my_delay_ms(17*(1+time),1,1,0);
}

void corner_cube (uint8_t time)
{
    uint8_t color[3]={0}, ct=0, n=0, hue=0, x1 = 0, x2=0, y1
= 0, y2=0, z1 = 0, z2=0;
    static uint8_t corner[2][3]={{0}};
          hue = rand() % 43;
    color[0]=pgm_read_byte(&redData[hue]);
    color[1]=pgm_read_byte(&greenData[hue]);
    color[2]=pgm_read_byte(&blueData[hue]);

    for (ct=0;ct<3;ct++)
    {
                    corner[0][ct]=corner[1][ct];
        corner[1][ct]=(rand() % 2)*4;
    }

          for(n = 0; n<2;n++)
          {
          for(ct = 0; ct<5;ct++)
          {
                            if(corner[n][0] == 0)
                            {
                                    x1 = 0;
                                    if(n==0)
                                            x2 = ct;
                                    else
                                            x2 = 4-ct;
                            }else{
                                    x2 = 4;
                                    if(n==0)
                                            x1 = 4-ct;
                                    else
                                            x1 = ct;
                            }
```

```
                        if(corner[n][1] == 0)
                        {
                                y1 = 0;
                                if(n==0)
                                        y2 = ct;
                                else
                                        y2 = 4-ct;
                        }else{
                                y2 = 4;
                                if(n==0)
                                        y1 = 4-ct;
                                else
                                        y1 = ct;
                        }
                        if(corner[n][2] == 0)
                        {
                                z1 = 0;
                                if(n==0)
                                        z2 = ct;
                                else
                                        z2 = 4-ct;
                        }else{
                                z2 = 4;
                                if(n==0)
                                        z1 = 4-ct;
                                else
                                        z1 = ct;
                        }
                        clear();
                        for (uint8_t x=x1;x<=x2;x++)
                {
                for (uint8_t y=y1;y<=y2;y++)
                {
                        for (uint8_t z=z1;z<=z2;z++)
                        {

if((x == x1 || x == x2) && (y == y1 || y == y2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

if((x == x1 || x == x2) && (z == z1 || z == z2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

if((y == y1 || y == y2) && (z == z1 || z == z2))

{

          put_XYZ(x,y,z,color[0],color[1],color[2]);

}

                                                }
                                        }
                        }
                if(my_delay_ms(17*(1+time),1,1,0))
                        {
                                        for (ct=0;ct<3;ct++)
                                    {
                                        corner[1][ct]=(rand()
% 2)*4;
                                    }
                                        return;
                        }
          }

          }
          if(my_delay_ms(17*(1+time),1,1,0))
          {
                    for (ct=0;ct<3;ct++)
              {
                  corner[1][ct]=(rand() % 2)*4;
                }
          }
}

void plasma (uint8_t time)
{
    uint8_t lx=0,ly=0,lz=0,vr=0,vg=0,vb=0;
          int r=0,g=0,b=0;
    static uint16_t radian_r=0,radian_g=0,radian_b=0;
    radian_r+=2;
    if (radian_r>=314)
          {
        radian_r-=314;
          }
    radian_g+=6;
    if (radian_g>=300)
          {
        radian_g-=300;
          }
    radian_b+=7;
    if (radian_b>=300)
    {
              radian_b-=300;
          }
    for (lx=0;lx<5;lx++)
    {
          for (ly=0;ly<5;ly++)
        {
          for (lz=0;lz<5;lz++)
            {
                                        r = (radian_r+20*
(lx-ly+lz));
                                        if (r>=314)
                                        {
                                    r-=314;
                                        }else if(r<0){
                                                r+=314;
                                        }
                                        g = (radian_g+20*
(ly-lz+lx));
                                        if (g>=300)
                                        {
                                    g-=300;
                                        }else if(g<0){
                                                g+=300;
                                        }
                                        b = (radian_b+20*
(lz-lx+ly));
                                        if (b>=300)
                                        {
                                    b-=300;
                                        }else if(b<0){
                                                b+=300;
                                        }
                                        vr = 7-pgm_read_byte
(&mysin[r]);
                                        vg = 7-pgm_read_byte
(&mysin[g]);
                                        vb = 7-pgm_read_byte
(&mysin[b]);
                                        put_XYZ
(lx,ly,lz,vr,vg,vb);
              }
          }
    }
    my_delay_ms(2*(3+time),1,1,0);
}

void chaos(uint8_t time)
{
           uint8_t hue=0;
           for (uint8_t x=0;x<5;x++)
    {
        for (uint8_t y=0;y<5;y++)
        {
            for (uint8_t z=0;z<5;z++)
            {
                                        hue = rand() % 43;
                                        put_XYZ(x,y,z,
```

```
          pgm_read_byte(&redData[hue]),

          pgm_read_byte(&greenData[hue]),

          pgm_read_byte(&blueData[hue]));
                              }
                    }
          }
          my_delay_ms(26*(1+time),1,1,0);
}

void cube_expand(uint8_t time)
{
          uint8_t hue=0;
          hue = rand() % 43;
          for(uint8_t n=0;n<2;n++)
          {
                    for(uint8_t diff=0;diff<3;diff++)
                    {
                              for (uint8_t x=2-
diff;x<=2+diff;x++)
                    {
                    for (uint8_t y=2-diff;y<=2+diff;y++)
                    {
                              for (uint8_t z=2-
diff;z<=2+diff;z++)
                              {

if(n == 0)

{



          put_XYZ(x,y,z,

                    pgm_read_byte(&redData[hue]),

                    pgm_read_byte(&greenData[hue]),

                    pgm_read_byte(&blueData[hue]));


}else{

          put_XYZ(x,y,z,0,0,0);

}
                                                  }
                                        }
                              }
                              if(my_delay_ms(25*(1+time),
1,1,0))
                              {
                                        return;
                              }
                    }
                    if(my_delay_ms(50*(1+time),1,1,0))
                    {
                              return;
                    }
          }
}

void fire(uint8_t c, uint8_t time)
{
    static uint8_t col2[5][5]={{0}};
    uint8_t set_x=0,set_z=0,rnd=0,r=0,g=0,b=0,max=0;
          r=pgm_read_byte(&redData[c]);
    g=pgm_read_byte(&greenData[c]);
    b=pgm_read_byte(&blueData[c]);
          for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                              rnd = rand()%7;
                              if(rnd == 0)
                              {
                                        if((set_x == 0 ||
set_x == 4) || (set_z == 0 || set_z == 4))
                                        {
                                                  max = 2;
                                        } else if((set_x ==
1 || set_x == 3) || (set_z == 1 || set_z == 3))
                                        {
                                                  max = 4;
                                        } else
                                        {
                                                  max = 5;
                                        }
                                        if(col2[set_x]
[set_z]!=max)
                                        {
                                                  col2
[set_x][set_z]++;
                                        }
                              }else if(rnd == 6)
                              {
                                        if(col2[set_x]
[set_z]!=0)
                                        {
                                                  col2
[set_x][set_z]--;
                                        }
                              }
                    }
          }
          col2[0][0]=0;
          col2[0][4]=0;
          col2[4][0]=0;
          col2[4][4]=0;
    for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                              for(uint8_t y=0;y<5;y++)
                              {
                              if(col2[set_x][set_z] > y)
                                        {
                                                  put_XYZ
(set_x,y,set_z,r,g,b);
                                                  if((set_x
== 0 || set_x == 4) || (set_z == 0 || set_z == 4))
                                                  {

fade_XYZ(set_x,y,set_z,6,6,6);
                                                  } else if
((set_x == 1 || set_x == 3) || (set_z == 1 || set_z == 3))
                                                  {

fade_XYZ(set_x,y,set_z,3,3,3);
                                                  } else if
(y > 2)
                                                  {

fade_XYZ(set_x,y,set_z,y-1,y-1,y-1);
                                                  }

                                        }else
                                        {
                                                  fade_XYZ
(set_x,y,set_z,1,1,1);
                                        }
                              }
                    }
    }
    if(my_delay_ms(3*(1+time),1,1,0))
          {
                    for(set_x=0;set_x<5;set_x++)
              {
                  for (set_z=0;set_z<5;set_z++)
                  {
                                        col2[set_x][set_z]
=0;
                              }
                    }
          }
}

void display_colors (uint16_t len)
```

```
{
    uint8_t lx=0,ly=0,lz=0,r=0,g=0,b=0;
    clear();
    for (lx=0;lx<5;lx++)
    {
        for (ly=0;ly<5;ly++)
        {
            for (lz=0;lz<5;lz++)
            {
                                    if(lx == 0)
                                    {
                                            r = 0;
                                    }else
                                    {
                                            r =
2*lx-1;
                                    }
                                    if(ly == 0)
                                    {
                                            g = 0;
                                    }else
                                    {
                                            g =
2*ly-1;
                                    }
                                    if(lz == 0)
                                    {
                                            b = 0;
                                    }else
                                    {
                                            b =
2*lz-1;
                                    }
                put_XYZ(lx,ly,lz,r,g,b);
            }
        }
    }
    my_delay_ms(len,0,0,0);
}

void airRaidReset(void)
{
          uint8_t set_x=0,set_z=0;
          for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                            col[set_x][set_z][0]=5;
             }
          }
}

void wormsReset(void)
{
          for(uint8_t m = 0; m < 3;m++)
          {
                    for(uint8_t n = 0; n < 4;n++)
                    {
                              worms[m][n][0]=2;
                              worms[m][n][1]=2;
                              worms[m][n][2]=2;
                    }
          }
}

uint8_t airRaid(uint8_t x, uint8_t z)
{
    static uint8_t cnt=0,s=0,t=0;
    uint8_t set_x=0,set_z=0;

    if(col[x][z][0]==4)
    {
                    if(col[x][z][1]==0)
                    {
                              clear();
                              for(set_x=0;set_x<5;set_x++)
                    {
                    for (set_z=0;set_z<5;set_z++)
                    {
                                                                        if(col
[set_x][set_z][0] < 5)
                                                                        {
                                                                        single_XYZ
(set_x,4-col[set_x][set_z][0],set_z,2*col[set_x][set_z][1],
7);
                                                                        }
                                        }
                    }
                                        put_XYZ (x,0,z,7,7,0);
                                        gameOver();
                                        s=0;
                                        t=0;
                                        airRaidReset();
                                        return 0;
                    }else
                    {
                                        if(s < 5)
                                        {
                                                  s = 0;
                                        }else
                                        {
                                                  s -= 5;
                                        }
                    }
    }

    if(cnt != 4)
    {
                    cnt++;
    }else
    {
                    for(set_x=0;set_x<5;set_x++)
          {
          for (set_z=0;set_z<5;set_z++)
          {
                                        col[set_x][set_z]
[0]++;
                    }
                    }
                    if (col[set_x = rand() % 5][set_z = rand
() % 5][0]>4)
          {
          col[set_x][set_z][0]=0;
                                        if(rand()%100 <= 3)
                                        {
                                                  col[set_x][set_z][1]
=1;
                                        }else
                                        {
                                                  col[set_x][set_z][1]
=0;
                                        }
          }
                    cnt=0;
    }

          if(t != 13)
    {
                    t++;
    }else
    {
                    if(s<30)
                    {
                              s++;
                    }
                    t=0;
    }

          clear();
    for(set_x=0;set_x<5;set_x++)
    {
        for (set_z=0;set_z<5;set_z++)
        {
                                        if(col[set_x][set_z][0] < 5){
                                        single_XYZ (set_x,4-col[set_x]
[set_z][0],set_z,2*col[set_x][set_z][1],7);
                                        }
                    }
    }
```

```
    single_XYZ (x,0,z,1,7);
    if(my_delay_ms(50-s,0,1,0))
          {
                    airRaidReset();
                    s=0;
                    t=0;
                    return 0;
          }
    return 1;
}

uint8_t Snake(uint8_t dir)
{
    static uint8_t snake[24][3]={{1,2,2}}, apple[3]={1,2,2},
len=0, crt=0;
          uint8_t n=0,x=0,y=0,z=0;

          void reset(void)
          {
                    len = 0;
                    crt = 10;
                    snake[0][0]=1;
                    snake[0][1]=2;
                    snake[0][2]=2;
                    apple[0]=1;
                    apple[1]=2;
                    apple[2]=2;
          }

          uint8_t colission()
          {
                    if(getColor(snake[0][0],snake[0]
[1],snake[0][2],1)==0)
                    {
                              return 0;
                    }
                    return 1;
          }

          uint8_t gotApple()
          {
                    if(snake[0][0]==apple[0] && snake[0][1]
==apple[1] && snake[0][2]==apple[2])
                    {
                              return 1;
                    }
                    return 0;
          }

          if(gotApple() == 1)
          {
                    while(getColor(x = rand() % 5,y = rand()
% 5,z = rand() % 5,2)!=0)
                    {
                    }
                    apple[0]=x;
                    apple[1]=y;
                    apple[2]=z;
                    if(len != 23)
                    {
                    len++;
                    }
          }

    if(crt != 50)
    {
                    crt++;
    }else
    {
                    if(len != 23)
                    {
                    len++;
                    }
                    crt = 0;
    }

    for(n=len;n>0;n--)
    {
                    snake[n][0]=snake[n-1][0];
                    snake[n][1]=snake[n-1][1];
                    snake[n][2]=snake[n-1][2];
    }

    switch (dir){
          case 0:
                              if(snake[0][0]==4)
                              {
                              snake[0][0]=0;
                              }else
                              {
                              snake[0][0]++;
                              }
                    break;
          case 1:
                              if(snake[0][2]==4)
                              {
                              snake[0][2]=0;
                              }else
                              {
                              snake[0][2]++;
                              }
                    break;
          case 2:
                              if(snake[0][0]==0)
                              {
                                  snake[0][0]=4;
                              }else
                              {
                                        snake[0][0]--;
                              }
                    break;
          case 3:
                              if(snake[0][2]==0)
                              {
                              snake[0][2]=4;
                              }else
                              {
                                  snake[0][2]--;
                              }
                    break;
          case 4:
                              if(snake[0][1]==4)
                              {
                                        snake[0][1]=0;
                              }else
                              {
                              snake[0][1]++;
                              }
                    break;
          case 5:
                              if(snake[0][1]==0)
                              {
                              snake[0][1]=4;
                              }else
                              {
                              snake[0][1]--;
                              }
                    break;
    }

    if(colission() == 1){
                    gameOver();
                    reset();
                    return 0;
    }

          clear();

          put_XYZ(apple[0],apple[1],apple[2],7,0,0);
          put_XYZ(snake[0][0],snake[0][1],snake[0][2],
0,7,0);

          for(n=1;n<=len;n++)
          {

                    put_XYZ(snake[n][0],snake[n][1],snake[n]
[2],7,7,0);

          }
```

```
	if(my_delay_ms(250,0,1,1))
	{
		reset();
		return 0;
	}
	return 1;

}

void gameOver(void)
{
	my_delay_ms(500,0,0,0);
	blackout(100);
	display_colors(500);
	blackout(100);

}
```

TEST_PATTERNS.C

```
#include "patterns.h"
#include "driver.h"
#include "test_patterns.h"
#include "uart001.h"


static uint8_t mode=0, time=6, color=0, snake=0, x=2, z=2;


int main(void)
{
    shiftReg_Setup();
	timerSetup();
	uartSetup();
    buttonSetup();
	wormsReset();
	airRaidReset();

	display_colors(1000);
    blackout(260);

	while(1)
    {
		//Checks if UART changes mode
		if (USART_DataRx())
		{
		char command=USART_Receive(); // buffer
the received character.
		switch (command) {
			case 'x':
				initMessage();
			break;
			case 'a':
				mode=0;
				uart_putsl_p(PSTR
("Flashing random"));
			blackout(260);
			break;
			case 'b':
				mode=1;
				uart_putsl_p(PSTR
("Chaos"));
			blackout(260);
			break;
			case 'c':
				mode=2;
				uart_putsl_p(PSTR
("Plasma"));
			blackout(260);
			break;
			case 'd':
				mode=3;
				uart_putsl_p(PSTR
("Corner Expanding"));
			blackout(260);

	break;
case 'e':
	mode=4;
	uart_putsl_p(PSTR
("Cubistic Expand 1"));
blackout(260);
break;
case 'f':
	mode=5;
	uart_putsl_p(PSTR
("Cubistic Expand 2"));
blackout(260);
break;
case 'g':
	mode=6;
	uart_putsl_p(PSTR
("Worm 1"));
blackout(260);
break;
case 'h':
	mode=7;
	uart_putsl_p(PSTR
("Worm 2"));
blackout(260);
break;
case 'i':
	mode=8;
	uart_putsl_p(PSTR
("Rain 1"));
blackout(260);
break;
case 'j':
	mode=9;
	uart_putsl_p(PSTR
("Rain 2"));
blackout(260);
break;
case 'k':
	mode=10;
	uart_putsl_p(PSTR
("Rain 3"));
blackout(260);
break;
case 'l':
	mode=11;
	uart_putsl_p(PSTR
("Rain 4"));
blackout(260);
break;
case 'm':
	mode=12;
	uart_putsl_p(PSTR
("Star 1"));
blackout(260);
break;
case 'n':
	mode = 13;
	uart_putsl_p(PSTR
("Star 2"));
blackout(260);
break;
case 'o':
	mode = 14;
	uart_putsl_p(PSTR
("Fire"));
blackout(260);
break;
case 'p': //speed +
		if(time>0)
{
time-=3;
		}
break;
case 'q': //speed -
	if(time<20){
time+=3;
		}
break;
case 'r': //color +
```

```
                                color=(color+1)%43;
                        break;
                        case 's': //color -
                                if(color == 0){
                                        color=41;
                                } else{
                                        color--;
                                }
                        break;

                        }
                }

        switch (mode){
                case 0:
                flash_rand(time);
                break;
                case 1:
                        chaos(time);
                        break;
                case 2:
                plasma(time);
                break;
                case 3:
                corner_expand(time);
                break;
                case 4:
                corner_cube(time);
                break;
                case 5:
                        cube_expand(time);
                        break;
                case 6:
                worm(0,time);
                break;
                case 7:
                worm(1,time);
                break;
                case 8:
                rain(0,0,color,time);
                break;
                case 9:
                rain(1,0,color,time);
                break;
                case 10:
                rain(0,1,color,time);
                break;
                case 11:
                rain(1,1,color,time);
                break;
                case 12:
                star(0,color,time);
                break;
                case 13:
                star(1,color,time);
                break;
                case 14:
                fire(color,time);
                break;
                case 15:
                if(Snake(snake)==0)
                        {
                                snake = 0;
                }
                break;
                case 16:
                if(airRaid(x,z))
                {
                        setX();
                        setZ();
                }else{
                                x = 2;
                        z = 2;
                }
                break;
                }

    }
}
```

```
void uartSetup(){

        USART_Init(UART_BAUD_SELECT
(9600,FCPU),USART_SET_8_1_N);   //baudrate = 9600 bps 8 data
bits 1 stop bit no parity
    sei(); // enable interupts
        uart_putsl_p(PSTR("*****************LOOPBACK-
RECONNECT****************"));
        initMessage();

}

void buttonSetup(){
        DDRD &= ~(UP|DOWN|LEFT|RIGHT|B1|B2);
        PORTD |= (UP|DOWN|LEFT|RIGHT|B1|B2); //Enable
pullup resitor on inputs
}

void initMessage(){
        // into string
---------------------------------------------
        uart_putsl_p(PSTR
("*******************************************"));
        uart_putsl_p(PSTR("LED CUBE SERIAL COMMUNICATION
INITIATED"));
        uart_putsl_p(PSTR("use under your own supervision
"));
        uart_putsl_p(PSTR("Created by: Anders, David,
Stefan"));
        uart_putsl_p(PSTR("Telnet commands are:
a,b,c,d,e,f,g,h,i,j,k,l,m,n.*"));
        uart_putsl_p(PSTR
("*******************************************"));
        // into string
---------------------------------------------
}

uint8_t getButton(uint8_t in){ //function that returns 1 if
the button is pressed (but not hold), otherwise 0

        static uint8_t d=0,u=0,l=0,r=0,b1=0,b2=0;
        switch (in) {
                case BU: //up
                        if((PIND & UP) == 0)
                        {
                                if(u == 0)
                                {
                                        u = 1;
                                        return 1;
                                }
                        }else
                        {
                                u = 0;
                        }
                break;
                case BD: //down
                        if((PIND & DOWN) == 0)
                        {
                                if(d == 0)
                                {
                                        d = 1;
                                        return 1;
                                }
                        }else
                        {
                                d = 0;
                        }
                break;
                case BL: //left
                        if((PIND & LEFT) == 0)
                        {
                                if(l == 0)
                                {
                                        l = 1;
                                        return 1;
                                }
                        }else
```

```
                {
                        l = 0;
                }
            break;
            case BR: //right
                if((PIND & RIGHT) == 0)
                {
                        if(r == 0)
                        {
                                r = 1;
                                return 1;
                        }
                }else
                {
                        r = 0;
                }
            break;
            case BB1: //button 1
                if((PIND & B1) == 0)
                {
                        if(b1 == 0)
                        {
                                b1 = 1;
                                return 1;
                        }
                }else
                {
                        b1 = 0;
                }
            break;
            case BB2: //button 2
                if((PIND & B2) == 0)
                {
                        if(b2 == 0)
                        {
                                b2 = 1;
                                return 1;
                        }
                }else
                {
                        b2 = 0;
                }
            break;
        }
        return 0;
}

uint8_t setMode(void)
{
    uint8_t down=0, up=0,m=0;
    m=mode;
    down = getButton(BB2);
    up = getButton(BB1);
        if(mode == 0 && down == 1)
        {
            mode=16;
        }else
        {
                mode-=down;
        }
        if(mode == 16 && up == 1)
        {
            mode=0;
        }else
        {
            mode+=up;
        }
    if(mode != m){
                blackout(260);
                return 1;
    }
    return 0;
}

void setTime(void)
{
    uint8_t down=0, up=0;
    down = getButton(BU);
    up = getButton(BD);
        if(time == 0 && down == 1){
            time=0;
        }else{
            time-=3*down;
        }
        if(time == 20 && up == 1){
            time=20;
        }else{
            time+=3*up;
        }
}

void setColor(void)
{
    uint8_t down=0, up=0;

    down = getButton(BL);
    up = getButton(BR);
        if(color == 0 && down == 1)
        {
            color=42;
        }else
        {
            color-=down;
        }
        if(color == 42 && up == 1)
        {
                color=0;
        }else
        {
                color+=up;
    }
}

void snakeDir(void)
{
    uint8_t down=0,up=0,left=0,right=0;
    down = getButton(BD);
    up = getButton(BU);
    left = getButton(BL);
    right = getButton(BR);
        if(up == 1)
        {
                        switch( snake ) {
                case 0:
                                                snake = 4;
                                                break;
                    case 1:
                                                snake = 4;
                                                break;
                    case 2:
                                                snake = 4;
                                                break;
                    case 3:
                                                snake = 4;
                                                break;
                    case 4:
                                                snake = 1;
                                                break;
                            case 5:
                                                snake = 1;
                                                break;
                        }
    }
    else if(left == 1)
        {
                switch( snake ) {
                case 0:
                                                snake = 1;
                                                break;
                    case 1:
                                                snake = 2;
                                                break;
                    case 2:
                                                snake = 3;
                                                break;
                    case 3:
                                                snake = 0;
                                                break;
                            case 4:
                                                snake = 2;
```

```
                        case 5:
                                                        break;

                                                        snake = 2;
                                                        break;

                                }
        }
    else if(down == 1)
        {
                switch( snake ) {
                case 0:
                                                        snake = 5;
                                                        break;

                case 1:
                                                        snake = 5;
                                                        break;

                case 2:
                                                        snake = 5;
                                                        break;

                case 3:
                                                        snake = 5;
                                                        break;

                case 4:
                                                        snake = 3;
                                                        break;

                case 5:
                                                        snake = 3;
                                                        break;

                                }
        }
    else if(right == 1)
        {
                switch( snake ) {
                case 0:
                                                        snake = 3;
                                                        break;

                case 1:
                                                        snake = 0;
                                                        break;

                case 2:
                                                        snake = 1;
                                                        break;

                case 3:
                                                        snake = 2;
                                                        break;

                case 4:
                                                        snake = 0;
                                                        break;

                case 5:
                                                        snake = 0;
                                                        break;

                                }
        }
}

void setX(void)
{
        uint8_t down=0, up=0;
    down = getButton(BL);
    up = getButton(BR);
        if(x == 0 && down == 1){
                x=0;
        }else if(x == 4 && up == 1){
                x=4;
        }else{
                x += up-down;
        }
}

void setZ(void)
{
    uint8_t down=0, up=0;
    down = getButton(BD);
    up = getButton(BU);
        if(z == 0 && down == 1){
                z=0;
        }else if(z == 4 && up == 1){
                z=4;
        }else{
                z += up-down;
        }
}



uint8_t my_delay_ms(uint16_t time, uint8_t tc, uint8_t m,
uint8_t s) // Delay in milliseconds
{
    while(time)
    {
                if(tc == 1)
                {
                        setTime();
                        setColor();
                }
                if(s == 1)
                {
                        snakeDir();
                }
                _delay_ms(1);
                time--;
                if(m == 1)
                {
                        if(setMode())
                        {
                                return 1;
                        }
                }
    }
        return 0;
}
```

UART001.H

```
#include "uart001.inc"

//        Prototypes
void USART_Init( uint16_t baudrate,uint8_t setup);
uint8_t USART_Receive( void );
void USART_Transmit(uint8_t data );
uint8_t USART_DataRx( void );
void uart_puts_p(const char *progmem_s );
void uart_putsl_p(const char *progmem_s );
uint8_t USART_Receive_uc( void );
uint8_t USART_Receive_lc( void );
```

UART001.C

```
// Based on ap note AVR306 by ATMEL
// Routines for interrupt controlled USART

// Last modified: 26 October 2006
// Changed to work with WinAVR 20060421

// Changed : 6 December 2005
// adderd uppercase and lowercase receive routines

// Changed : 3 Nov 2005
// changed rx buffer access, added checks for non-atomic
interrupt pointer modification
// changed function name DataInReceiveBuffer to USART_DataRx

// modified: 10 APR 2005
// added double speed baud support;
// Modified by: Murray Horn

// this file includes the tx and rx routines for a single
uart mega avr.
// seperate tx and rx ques are available and can be sized in
the .inc file
// a string tx feature from rom has been added
(uart_puts_p).

//        Includes
#include <inttypes.h>
#include <avr/interrupt.h>
```

```
#include <avr/pgmspace.h>     // used for the string
printing feature

#include "uart001.h"                    // Prototypes

// Static Variables
static uint8_t USART_RxBuf[USART_RX_BUFS];
static volatile uint8_t USART_RxHead;
static volatile uint8_t USART_RxTail;
static uint8_t USART_TxBuf[USART_TX_BUFS];
static volatile uint8_t USART_TxHead;
static volatile uint8_t USART_TxTail;


//-----------------------------------------------------
// Initialize USART
void USART_Init( uint16_t baudrate,uint8_t setup)
{
//------------------------------
// the doublespeed selector
          if (baudrate > 0x7ff)
          {
                    baudrate += 1;
                    baudrate = baudrate >> 1;
                    baudrate -= 1;
                    UCSRA = 0;
          }
          else
                    UCSRA = (1<<U2X);
//------------------------------

//                  Set the baud rate
    UBRRH = (uint8_t)(0x0f &(baudrate >> 8));
          UBRRL = (uint8_t) baudrate;


//                  Enable UART receiver and transmmitter
and receive int
    UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);

//                  Set frame format: data bits, stop
bits,parity etc
          UCSRC = (1<<URSEL)| setup ; // set the uart
bits,baudrate,stop bits

//                  Flush buffers
          USART_RxTail = 0;
          USART_RxHead = 0;
          USART_TxTail = 0;
          USART_TxHead = 0;

}
//-----------------------------------------------------


//-----------------------------------------------------
//        RX Interrupt handler
SIGNAL(SIG_UART_RECV)
{
          uint8_t data;
          uint8_t tmphead;
          uint8_t nxthead;

//        Read the received data
          data = UDR;
//        Calculate buffer index
          tmphead = USART_RxHead;
          nxthead = ( tmphead + 1 ) & USART_RX_BUFFER_MASK;

          if ( nxthead == USART_RxTail )
          {
// ERROR! Receive buffer overflow
          }
          else
          {
                    USART_RxBuf[tmphead] = data; // Store
received data in buffer
                    USART_RxHead = nxthead;      // Store
new index
          }
}
//-----------------------------------------------------


//-----------------------------------------------------
//        TX Interrupt handler
SIGNAL(SIG_UART_DATA)
{
          uint8_t tmptail;

//        Check if all data is transmitted
          if ( USART_TxHead != USART_TxTail )
          {
//        Calculate buffer index
                    tmptail = ( USART_TxTail + 1 ) &
USART_TX_BUFFER_MASK;
                    USART_TxTail =
tmptail;                    // Store new index

                    UDR = USART_TxBuf[tmptail];   // Start
transmition
          }
          else
          {
                    UCSRB &= ~
(1<<UDRIE);                         // Disable UDRE
interrupt
          }
}
//-----------------------------------------------------


//-----------------------------------------------------
//        RX function
uint8_t USART_Receive( void )
{
          uint8_t tmptail,tmphead;
          uint8_t b;

          do
                    {
                    tmptail =
USART_RxTail;
          //
// double get the head to ensure no interrupt based
corruption of the pointer
                    do
                              tmphead = USART_RxHead;
                    while (tmphead !=
USART_RxHead);               // Wait for incomming data
                    }
          while ( USART_RxHead ==
tmptail );                   // Wait for incomming data


          b = USART_RxBuf[tmptail];

          tmptail = ( tmptail + 1 ) &
USART_RX_BUFFER_MASK; // Calculate buffer index

          USART_RxTail =
tmptail;
// Store new index

          return b;                                 // Return
data
}
//-----------------------------------------------------

//-----------------------------------------------------
// get the received byte in lower case
uint8_t USART_Receive_lc( void )
{
          uint8_t   ch;
          ch = USART_Receive();
          if (('A' <= ch) && ('Z' >= ch))
            ch = ch - ('A'-'a');
          return ch;
}
```

```
//-------------------------------------------------------

//-------------------------------------------------------
// get the received byte in upper case
uint8_t USART_Receive_uc( void )
{
          uint8_t   ch;
          ch = USART_Receive();
          if (('a' <= ch) && ('z' >= ch))
            ch = ch + ('A'-'a');
          return ch;
}
//-------------------------------------------------------


//-------------------------------------------------------
//        TX function
void USART_Transmit( uint8_t data )
{
          uint8_t tmphead;
//        Calculate buffer index
          tmphead = ( USART_TxHead + 1 ) &
USART_TX_BUFFER_MASK;         // Wait for free space in
buffer
          while ( tmphead == USART_TxTail );

          USART_TxBuf[tmphead] =
data;                         // Store data in buffer
          USART_TxHead =
tmphead;                                          // Store
new index

          UCSRB |=
(1<<UDRIE);
//        Enable UDRE interrupt
}
//-------------------------------------------------------


//-------------------------------------------------------
uint8_t USART_DataRx( void )
{
          uint8_t   again;
          uint8_t tmptail,tmphead;

          do        {
                    again = 0;
                    tmphead = USART_RxHead;
                    tmptail = USART_RxTail;
                    if (tmphead != USART_RxHead) again = 1;
                    if (tmptail != USART_RxTail) again = 1;
                    }
          while (again);                // Wait for
incomming data

          if ( tmphead == tmptail )
                    return(0);
          else
                    return(1);
}
//-------------------------------------------------------


//-------------------------------------------------------
// write a string to the usart from the program memory
void uart_puts_p(const char *progmem_s )
{
register char c;
          while ( (c = pgm_read_byte(progmem_s++)) )
                    USART_Transmit(c);
}
//-------------------------------------------------------

//-------------------------------------------------------
// write a string to the usart from the program memory and
add char 13 and 10
void uart_putsl_p(const char *progmem_s )
{
register char c;
          while ( (c = pgm_read_byte(progmem_s++)) )
                    USART_Transmit(c);
 // new line
          USART_Transmit(0x0d);
          USART_Transmit(0x0a);
}
//-------------------------------------------------------
```

GUI.PY

```
#! /usr/bin/env python

import serial
import Tkinter
from Tkinter import *

def xit():
        ser.close()
        quit(window)

def button1():
        ##window.b1.configure(state=DISABLED,
background='cadetblue')
        ser.write('a')
        current.set(ser.read(17))

def button2():
        ser.write('b')
        current.set(ser.read(7))


def button3():
        ser.write('c')
        current.set(ser.read(8))


def button4():
        ser.write('d')
        current.set(ser.read(18))

def button5():
        ser.write('e')
        current.set(ser.read(19))

def button6():
        ser.write('f')
        current.set(ser.read(19))


def button7():
        ser.write('g')
        current.set(ser.read(8))

def button8():
        ser.write('h')
        current.set(ser.read(8))

def button9():
        ser.write('i')
        current.set(ser.read(8))


def button10():
        ser.write('j')
        current.set(ser.read(8))

def button11():
        ser.write('k')
        current.set(ser.read(8))

def button12():
        ser.write('l')
        current.set(ser.read(8))

def button13():
        ser.write('m')
        current.set(ser.read(8))

def button14():
```

```
        ser.write('n')
        current.set(ser.read(8))
def button15():
        ser.write('o')
        current.set(ser.read(6))

#speed and color buttons
def button17(): #speed +
        ser.write('p')

def button18(): #speed -
        ser.write('q')

def button19(): #color +
        ser.write('r')

def button20(): #color -
        ser.write('s')


class Visual:
        def __init__(self, master):
                message = Label(window, textvariable
=start).grid(row=0, column=0, columnspan=4,sticky="WENS");
                currentLabel=Label
(window,textvariable=current).grid(row=8,column=0,
columnspan=4,sticky="WENS");
                b1 = Button(window, text = 'Random flash',
command = button1)
                b1.grid(row=3,column=0,sticky="WENS")
                b2 = Button(window, text = 'Chaos', command
= button2)
                b2.grid(row=3,column=1,sticky="WENS")
                b3 = Button(window, text = 'Plasma', command
= button3)
                b3.grid(row=3,column=2,sticky="WENS")
                b4 = Button(window, text = 'Corner Expand',
command = button4)
                b4.grid(row=3,column=3,sticky="WENS")
                b5 = Button(window, text = 'Cubistic Expand
1', command = button5)
                b5.grid(row=4,column=0,sticky="WENS")
                b6 = Button(window, text = 'Cubistic Expand
2', command = button6)
                b6.grid(row=4,column=1,sticky="WENS")
                b7 = Button(window, text = 'Worm 1', command
= button7)
                b7.grid(row=4,column=2,sticky="WENS")
                b8 = Button(window, text = 'Worm 2', command
= button8)
                b8.grid(row=4,column=3,sticky="WENS")
                b9 = Button(window, text = 'Rain 1', command
= button9)
                b9.grid(row=5,column=0,sticky="WENS")
                b10 = Button(window, text = 'Rain 2',
command = button10)
                b10.grid(row=5,column=1,sticky="WENS")
                b11 = Button(window, text = 'Rain 3',
command = button11)
                b11.grid(row=5,column=2,sticky="WENS")
                b12 = Button(window, text = 'Rain 4',
command = button12)
                b12.grid(row=5,column=3,sticky="WENS")
                b13 = Button(window, text = 'Star 1',
command = button13)
                b13.grid(row=6,column=0,sticky="WENS")
                b14 = Button(window, text = 'Star 2',
command = button14)
                b14.grid(row=6,column=1,sticky="WENS")
                b15 = Button(window, text = 'Fire', command
= button15)
                b15.grid(row=6,column=2,sticky="WENS")
                b16 = Button(window, text = 'Cube Xit',
command = xit, bg="red")
                b16.grid(row=6,column=3,sticky="WENS")
                b17 = Button(window, text = 'Speed +',
command = button17)
                b17.grid(row=7,column=0,sticky="WENS")
                b18 = Button(window, text = 'Speed -',
command = button18)
                b18.grid(row=7,column=1,sticky="WENS")
                b19 = Button(window, text = 'Color +',
command = button19)
                b19.grid(row=7,column=2,sticky="WENS")
                b20 = Button(window, text = 'Color -',
command = button20)
                b20.grid(row=7,column=3,sticky="WENS")
                ##Highlighting of buttons


ser=serial.Serial('COM18', 9600)
window = Tk()      #i am the parent, button = child
current=StringVar()
current.set("START")

start = StringVar()
ser.write('x')
start.set(ser.read(256))
stuff=Visual(window)

start ="NO CONNECTION!"

window.title("LED CUBE INTERFACE 1.1") ## Set the title of
the window to current animation.
window.mainloop()
```

Bilaga LED, IO, Ström och Priskalkyl

Kub storlek	Antal LED i kub	Antal IO för att styra kub 1 färg	Antal IO för att styra kub RGB	Antal IO för att styra kub RGB en färg åt gången hela kuben	Ström för 1 lager, 1 färg (10mA per led) [mA]	Ström för 1 lager, RGB (10mA per led) [mA] (alla färger tända)	Ström för 1 lager, 1 färg (20mA per led) [mA]	Ström för 1 lager, RGB (20mA per led) [mA] (alla färger tända)	Antal skiftregister för att styra kolumner, 1 färg	Antal skiftregister för att styra kolumner, RGB	Pris uppskattning för shiftreg + LED + avr + motstånd + strömreg. + Experimentkort, 1 färg	Pris uppskattning för shiftreg + LED + avr + motstånd + strömreg. + Experimentkort, RGB	Kompo
2	8	6	14	9	40	120	80	240	1	2	107,57 kr	182,28 kr	
3	27	12	30	15	90	270	180	540	2	4	121,21 kr	205,31 kr	Shiftre
4	64	20	52	23	160	480	320	960	2	6	137,86 kr	231,04 kr	atmeg
5	125	30	80	33	250	750	500	1500	4	10	175,49 kr	270,55 kr	5mm g
6	216	42	114	45	360	1080	720	2160	5	14	221,53 kr	314,56 kr	5mm R
7	343	56	154	59	490	1470	980	2940	7	19	288,86 kr	369,06 kr	Experi
8	512	72	200	75	640	1920	1280	3840	8	24	370,00 kr	429,86 kr	MOSF
9	729	90	252	93	810	2430	1620	4860	11	31	482,92 kr	508,04 kr	Konde
10	1000	110	310	113	1000	3000	2000	6000	13	38	615,05 kr	594,32 kr	Motstå