School of Electronic and Electrical Engineering
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Arduino-based games system

David Spickett

MEng in Electronic and Electrical Engineering

ELEC2645 Embedded Systems Project

Session	2010 / 2011
Student ID	200483181

Contents

Abstract

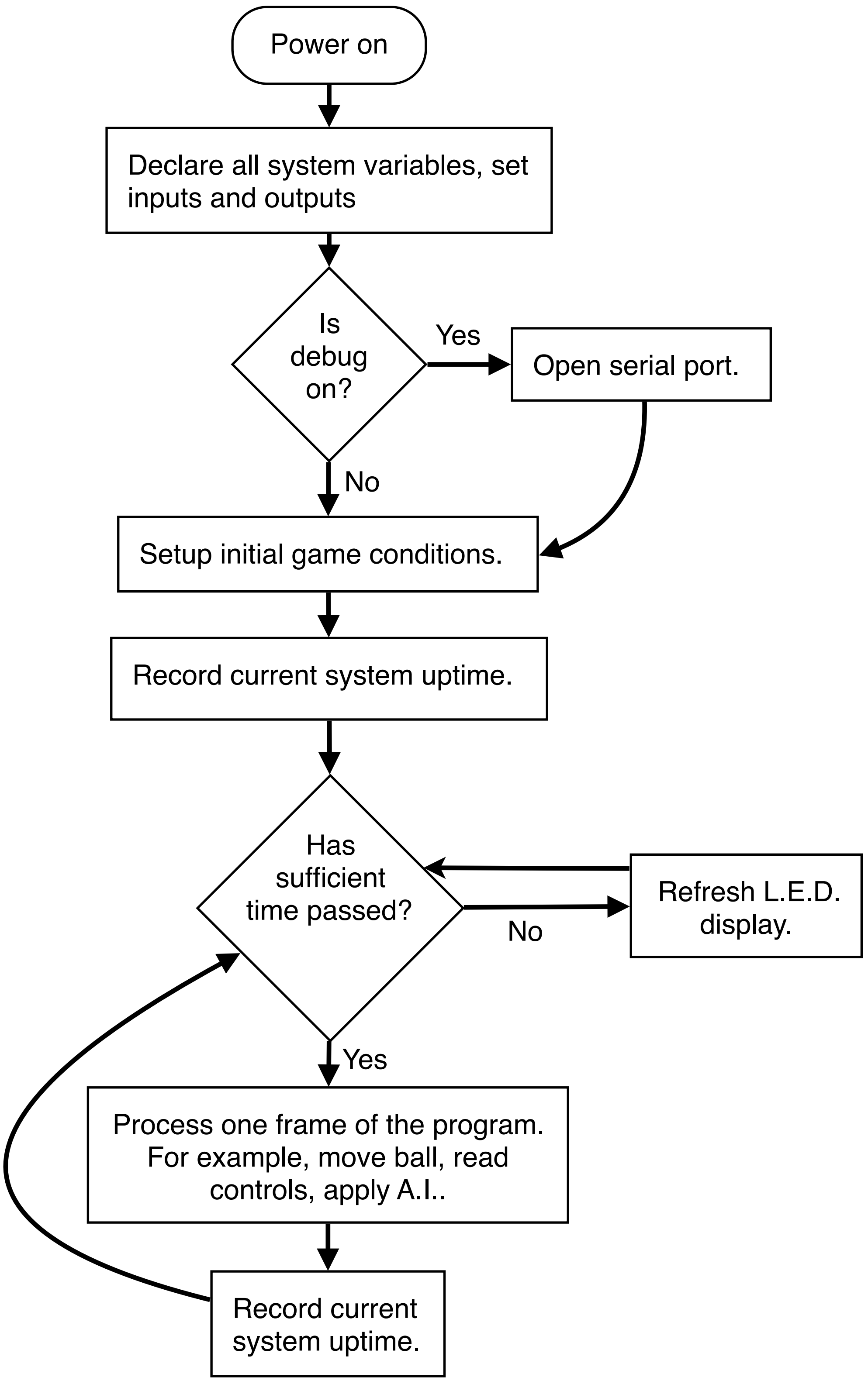
The following document details the design and construction of an Arduino based games system. It includes a display, buttons, external inputs and a speaker connection. It is designed to be reprogrammable and allow users to easily write applications making use of the hardware.

Introduction

The aim of this project was to make a platform to allow users to easily write and quickly iterate Arduino programs that make use of a display. Particularly interactive programs such as games. The system includes and L.E.D. display, input buttons, external inputs and audio output. The final goal is to have the system be much like a development board one might supply to developers prior to the production of a system, abstracting all the hardware access into simple code.

Software Overview

The software for the system is written as a template file which programs are built on. The flowchart below is for a generic program.


Power on
Declare all system variables, set inputs and outputs
Is debug on?
Yes
Open serial port.
No
Setup initial game conditions.
Record current system uptime.
Has sufficient time passed?
No
Refresh L.E.D. display.
Yes
Process one frame of the program. For example, move ball, read controls, apply A.I..
Record current system uptime.

The template has all of the pin numbers built into it by making use of #define statements. For example:

```
#define speakerPin 10
```

Using defines reduces program size, makes the built in functions easier to understand and the system configurable.

screenArray -
- a 10x7 array of bytes, which holds all the pixels which make up the screen.
- this is only ever accessed by the system itself.

modeSwitch -
- a byte which holds the current state of the mode switch.
- can be read by calling “readModeSwitch”.

controllerOne & controllerTwo -
- (each one is) a byte which holds the states of the inputs.
- is updated by calling readControllerOne/Two and then queried using readControllerOne/TwoButton(button).

dataValues -
- this is only ever used by the system and is an array of bytes, each being a power of 2.
- it is used to speed up other system functions by being a lookup table where the code would normally do a calculation.

time -
- an unsigned long which is used to hold the current system uptime in milliseconds.
- this is used to time when the loop of a given program will run, allowing the screen to be refreshed at all other times.

Finally there are functions built in to handle reading and writing to the various pieces of hardware.

setPixel(x,y) -
- turns on the pixel at x,y on the screen.

clearPixel(x,y) -
- as setPixel except the pixel at x,y is turned off.

readPixel(x,y) -
- returns the state of the pixel at x,y.

clearScreen() -
- turns off all pixels.

readControllerOne() -
- reads the built in buttons and store result in controllerOne.

readControllerTwo() -
- as for readControllerOne().

readControllerOneButton(button) -
- reads a given button and returns true if it is held down, false otherwise.

- this makes use of binary logic to do this. The value of the port is anded with the value of the button (in powers of 2), to isolate that bit and if the result is equal to the value of the button then it returns true.

readControllerTwoButton(button) -

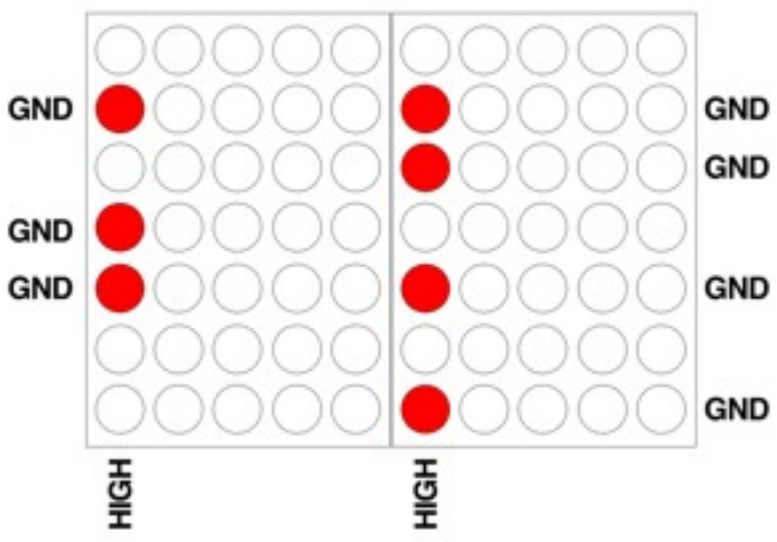
- as for readControllerOneButton(button).

playTone(frequency,duration) -

- plays a given frequency for a given time through the speaker output.

updateScreen() -

- this takes the current screenArray and applies it to the actual L.E.D. grids.
- it does this by scanning across both grids simultaneously, setting the same column on each one. Setting a row low turns on the L.E.D. in the current row and column.



This function calculates values based on each column of screenArray and outputs these using code based upon the Arduino's built in shiftOut function.

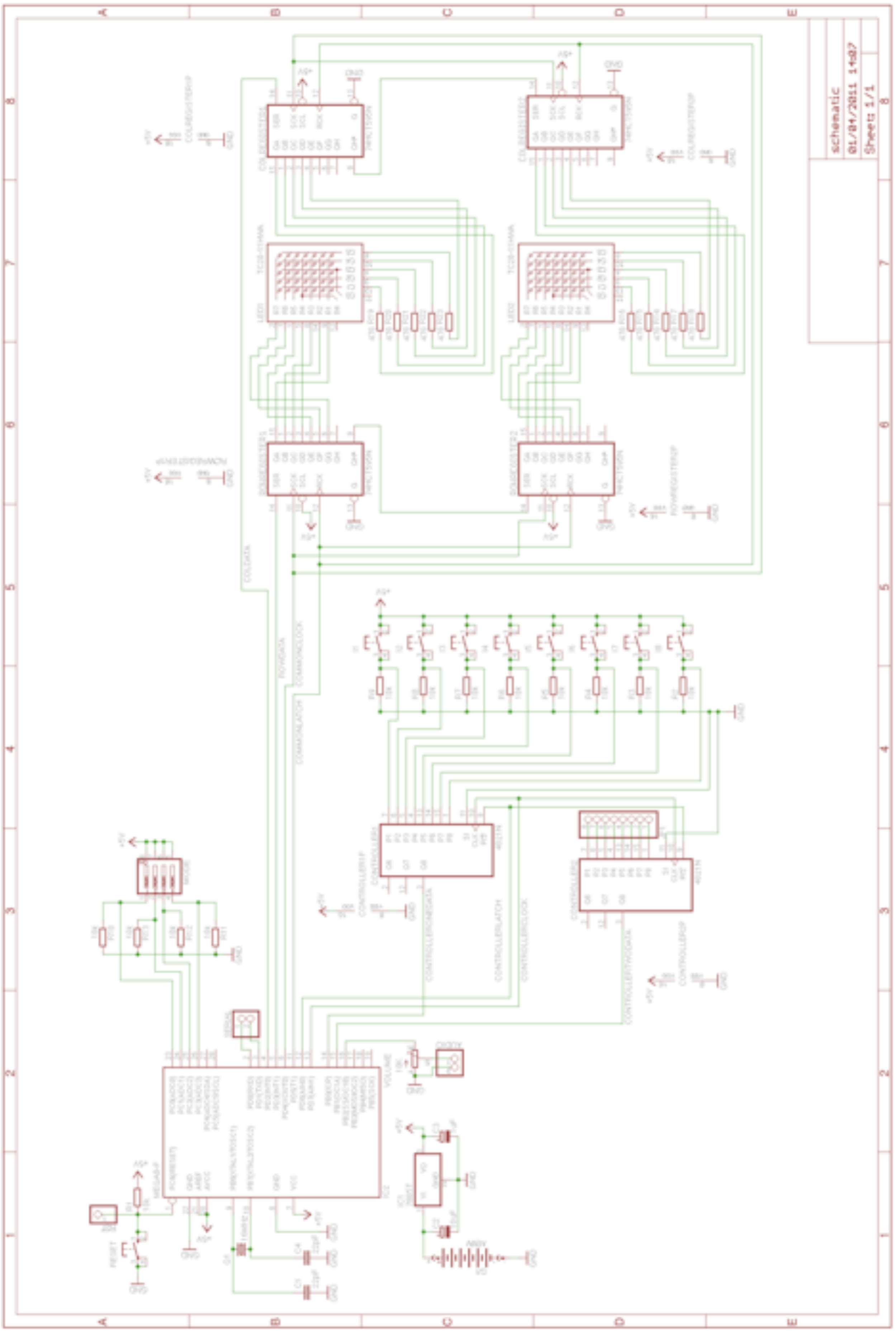
shiftIn(dataPin) -

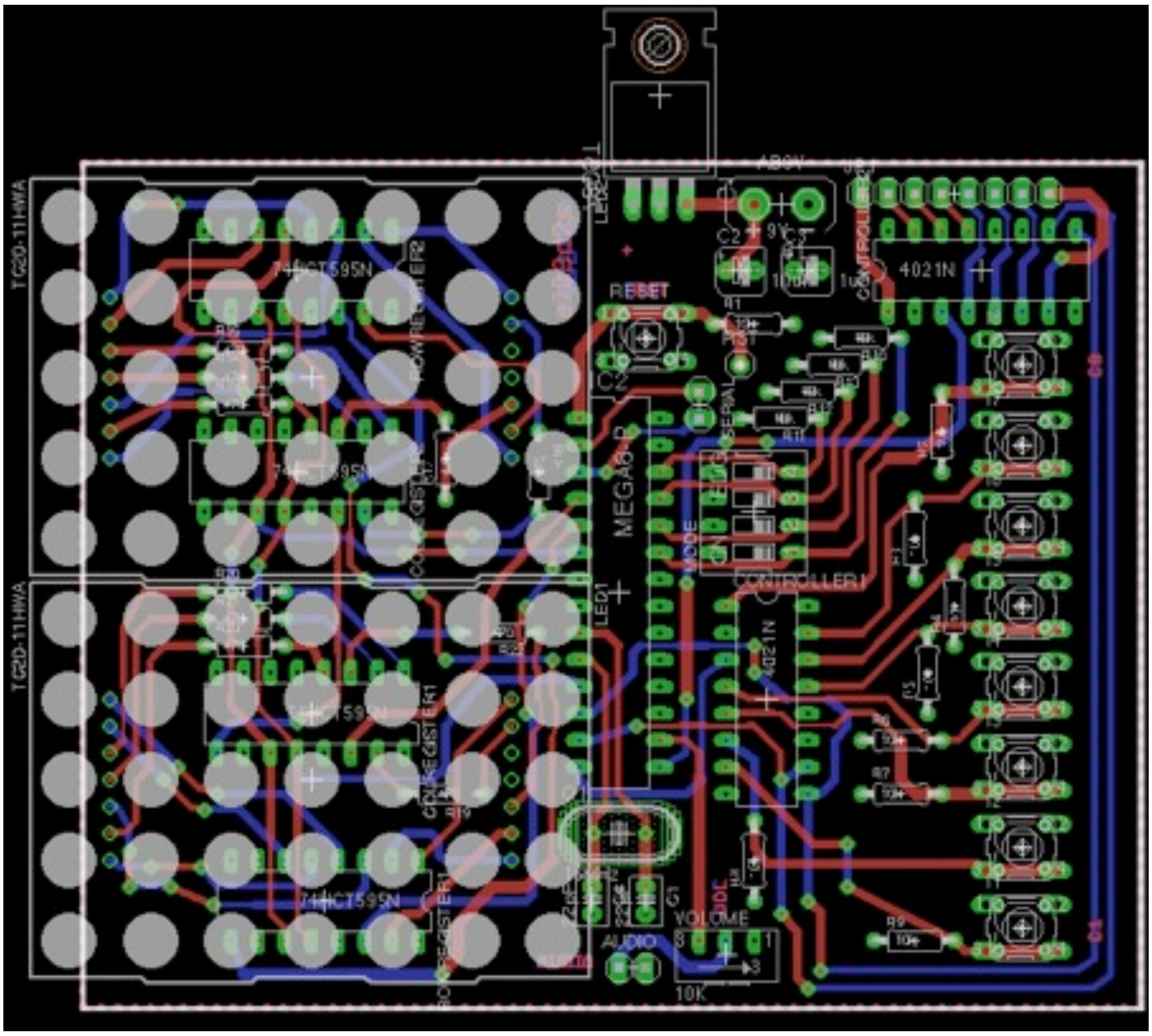
- this is based on source code from the Arduino shiftIn tutorials, with optmisations for this setup.

readModeSwitch() -

- this reads the value of the mode switch into "modeSwitch".

Hardware Overview





Part	Value	Device	Package	Library
AUDIO	n/a	PINHD-1X2	1X02	pinhead
C1	22pF	C2, 5-3	C2.5-3	capacitor-wima
C2	10uF	CPOL-EUB45181A	B45181A	resistor
C3	1uF	CPOL-EUB45181A	B45181A	resistor
C4	22pF	C2, 5-3	C2.5-3	capacitor-wima
COLREGISTER1	74HCT595N	74HCT595N	DIL16	74xx-eu
COLREGISTER2	74HCT595N	74HCT595N	DIL16	74xx-eu
CONTROLLER1	4021N	4021N	DIL16	40xx

Part	Value	Device	Package	Library
CONTROLLER 2	4021N	4021N	DIL16	40xx
G1	AB9V	AB9V	AB9V	battery
I1	n/a	10-XX	B3F-10XX	switch-omron
I2	n/a	10-XX	B3F-10XX	switch-omron
I3	n/a	10-XX	B3F-10XX	switch-omron
I4	n/a	10-XX	B3F-10XX	switch-omron
I5	n/a	10-XX	B3F-10XX	switch-omron
I6	n/a	10-XX	B3F-10XX	switch-omron
I7	n/a	10-XX	B3F-10XX	switch-omron
I8	n/a	10-XX	B3F-10XX	switch-omron
IC1	7805T	7805T	TO220H	linear
IC2	MEGA8-P	MEGA8-P	DIL28-3	atmel
JP1	n/a	PINHD-1X8	1X08	pinhead
LED1	TC20-11HWA	TC20-11HWA	TC20-11HWA	display-kingbright
LED2	TC20-11HWA	TC20-11HWA	TC20-11HWA	display-kingbright
MODE	n/a	SW_DIP-4	EDG-04	special
Q1	16MHz	CRYSTALHC49U70	HC49U70	crystal
R1	10k	R-EU_0204/7	0204/7	resistor
...	...	...	...	...
R13	10k	R-EU_0204/7	0204/7	resistor
R14	470	R-EU_0204/7	0204/7	resistor
...	...	...	...	...
R23	470	R-EU_0204/7	0204/7	resistor
RESET	n/a	10-XX	B3F-10XX	switch-omron
ROWREGISTER1	74HCT595N	74HCT595N	DIL16	74xx-eu

Part	Value	Device	Package	Library
ROWREGISTER2	74HCT595N	74HCT595N	DIL16	74xx-eu
RST	n/a	PINHD-1X1	1X01	pinhead
SERIAL	n/a	PINHD-1X2	1X02	pinhead
VOLUME	10k	TRIM_EU-S63S	S63S	pot

Most of the components are very basic but I will go into a few particular choices.

L.E.D. grids -

For this project I used 5x7 grids as they were readily available and combining two of them gives a display wider than it is high, which is sometimes advantageous.

Shift Registers -

There is one register for the first grid's rows and columns and the second's, for a total of four. Each pair of registers is tied together using the first's serial output. Every time the screen updates the system pushes two bytes to each pair of registers and the first byte overflows to the second. As this operation is synchronised, this means that all the shift clock and latch clocks can be controlled by the same lines. This reduces the outputs needed.

Testing

Hardware -

My final P.C.B. was missing the voltage regulator due to an error in my design. With the voltage regulator's footprint positioned over the edge of the design, it was excluded. This was only a minor set back as I can use an external supply to deliver the voltage needed or possibly make a custom cable that would include a battery clip and regulator in one.

I also confirmed that the micro-controller can be programmed whilst on the board by hooking it up to my own usb Arduino board.

Software -

I wrote a test program for the unit which does the following things:

- display the state of all inputs.
- display the state of the mode switch.
- output a constant tone to the audio port.
- run a display test pattern when certain buttons are held.

This showed that all the input hardware was correct (the mode switch was tested using external inputs as the component was wasted on the first P.C.B.) but that the display orientation was not. Initially the displays were in the wrong order, which I solved by changing the order data is sent, so as to invert and mirror the co-ordinates.

Conclusion

This project has impressed upon me the importance of making software flexible, in my case this has enabled me to correct what would otherwise have been show stopping hardware mistakes. I feel the design fits well onto it's current layout but if I were given the chance to improve it I would move the buttons to the sides of the screens, in two groups of four, like a conventional gamepad. Then I could add pull down resistors to the external inputs and possibly a header to the built in buttons to further expand the system's adaptability.

In the future I would like to expand on this design by using a larger, rgb display and a single micro-controller for driving it. This would allow the system to control the colour mix and brightness more accurately whilst separate controller deals with program logic and sends the driver pixel values over serial. Much like a conventional P.C. would with a discrete graphics card.

References

Arduino "wiring_shift.c" source code - http://code.google.com/p/arduino/source/browse/trunk/hardware/arduino/cores/arduino/wiring_shift.c (this was used as the basis for my functions)

Row-Column Scanning to control an 8x8 LED Matrix - http://www.arduino.cc/en/Tutorial/RowColumnScanning (background on the technique, although I did not use their code)

RCP Electronics EagleCad Tutorials - http://youtu.be/qG0O9LKH-_E (a series of videos detailing how to use Eagle)