iBantumi **David Piggott**

Contents

Project Outline	3	Mistakes in Artwork Set 0.2 - Player Two Upper	90
Initialisation Time Plan	4	PCB Pictures	91
Research - Existing Products	5	Pit Boards Smoke Test	94
Bantumi (Board Diagram)	6	Schematic & Artwork Development	108
Bantumi (Objective, Rules and Terminology)	7	Production Time Plan	112
Task Analysis	8	Proof of Concept Program	114
System Ideas	13	Component Research	123
Case Ideas	18	Case Development	127
Specification	24	Case Production	145
Development Time Plan	26	Case Pictures	150
Component Research	27	Time Usage & Unfortunate Circumstances	154
LED Control Protocol Ideas	43	Component Usage	158
LED Register Code Development - 0.1	44	Program Development	160
Problematic Programming	57	Testing	175
Development Dependencies	59	Evaluation	176
Schematic & Artwork Development	60	Industrial Practices, Systems & Control	186
Case Plan - Component Locations Draft	70	Bibliography & Conclusion	189
Schematic & Artwork Development	71	Appendix A - Rough notes	190
Pit Numbers, LED Mappings & Switch Array Addresses	79	Appendix B - Bao Idea & Redundant development prep.	233
LED Register Code Development - 0.2	80	Appendix C - Datasheets & Project files	240

Project Outline

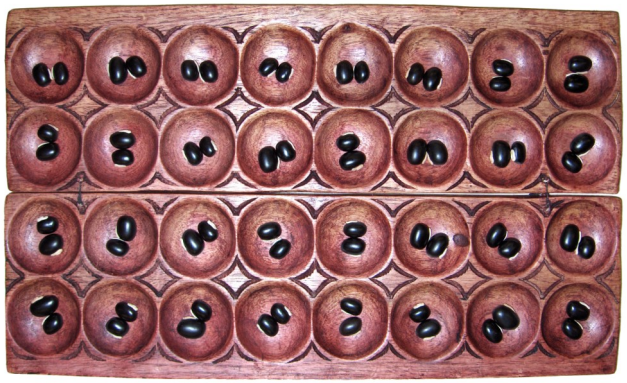


Figure 1.0

During the summer holidays I spent a month in Malawi on expedition with Team Challenge. While out there I became aware of a popular African board game. The game common in the areas we visited is known as Bao, but I was later to discover that it is one type of a family of games known collectively as mancala. Figure 1.0 is a picture of the Bao board I purchased at Chitimba beach camp on Lake Malawi.

I had decided that I would make an electronic version of Bao several months before the time came to start work on module SCT5, and had already thought quite a bit about how it would work. In doing so I realised that no matter how much I optimised the design, it would still be very complicated to develop and produce, so much so that I decided it not worth the risk, given the time constraints.

Not wanting to change the project to something entirely different, I decided to try and make an electronic version of one of the more simple mancala games, and recalled playing a game called 'Bantumi' on my Nokia 5510 some years ago. Unsure of the rules, I refreshed my memory by finding a PC software version, pictured in figure 1.1.

Figure 1.1

A description of the objectives and rules of Bao, along with all the ideas I had come up with, is included in appendix A. Included in the main documentation is only that which is relevant to the game of Bantumi - which I am going to make an electronic version of.

Initialisation Time Plan

ID	Task Name	Start	Finish	Duration
1	Christmas Holidays	15/12/2007	06/01/2008	23d
2	Spring Half Term	09/02/2008	17/02/2008	9d
3	Easter Holidays	20/03/2008	06/04/2008	18d
4	Introductory documentation	01/11/2007	01/11/2007	1d
5	Explanation of idea	02/11/2007	03/11/2007	2d
6	Analysis of idea	03/11/2007	04/11/2007	2d
7	Task Analysis	07/11/2007	09/11/2007	3d
8	Research	01/11/2007	01/12/2007	31d
9	Specification	09/11/2007	09/11/2007	1d
10	Development time plan	10/11/2007	10/11/2007	1d
11	Development	11/11/2007	27/01/2008	78d
12	Production time plan	28/01/2008	28/01/2008	1d
13	Production	29/01/2008	12/03/2008	44d
14	Completion of documentation	13/03/2008	19/03/2008	7d

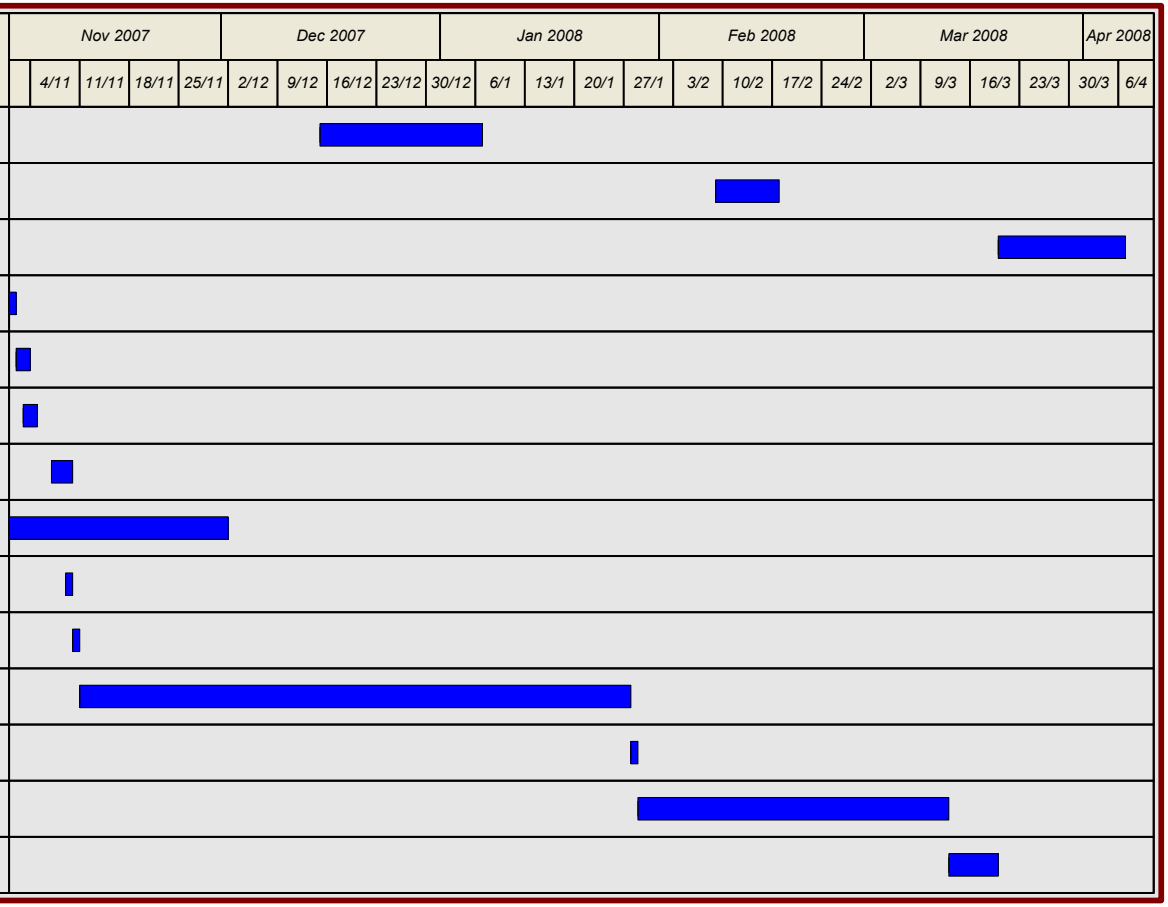


Figure 2.0 - Initialisation time plan

Research - Existing Products

Figure 3.0 - Playing Bao

The Mancala games available fall into two categories:

- Software games
- Wooden board games

The software games are typically free downloads for PCs, though there are also versions for embedded devices such as some older Nokia mobile phones. A phone I once had, the Nokia 5510 had a mancala game known as Bantumi built into it.

The wooden board games are not easily available in Europe and are most easily found in Africa. The Bao board I purchased was 1400 Kwacha (approximately $10/£5).

The available boards vary significantly in size and type. There are many different versions of the rules and therefore different boards to accommodate this. Different sizes are available depending on where the board will be used.

See figure 3.0 for an example of this; the big board being played on belonged to the beach resort we were staying at and therefore didn't need to be portable, whereas the board I bought is aimed at tourists and so was smaller to make it easier to bring back.

It is my hope that the board I produce will be smaller still than the portable board I bought.

Bantumi (Board Diagram)

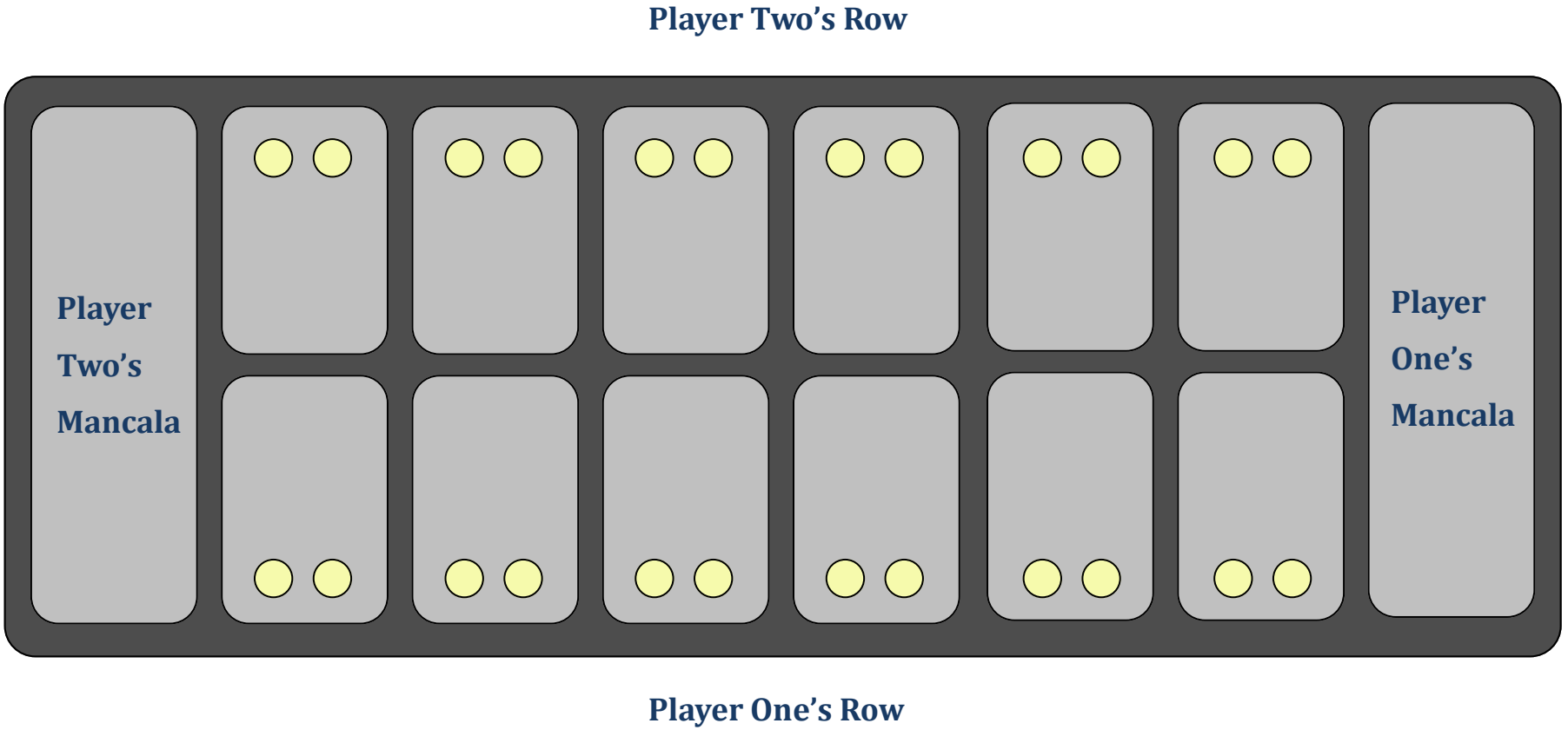


Figure 4.0 - Bantumi (Board Diagram)

Bantumi (Objective, Rules and Terminology)

Terms

Pits - The hollows in the board in which seeds are placed (there are six per player in Bantumi).
Seeds - The pieces that the game are played with. In Bantumi, the starting number varies, typically two or four.
Rows - A row of pits. In Bantumi there is only one row per player.
Mancala - The pit to the right of the players row, in which they must accumulate more seeds than their opponent does in theirs.

Objective

The game ends when one player can no longer make any moves (i.e. their row is empty), and the winner is the player with the most seeds in their mancala.

Rules

A players turn may consist of one or more moves. The following rules describe how to move the seeds:

1. Play takes place anticlockwise only
2. A turn begins when the player chooses a pit to initialise play with. The seeds of that pit are taken into the hand and sown along the board, moving towards their mancala (to the right of their row), depositing one seed in each pit passed until the hand is empty.
3. If there are still seeds in the hand after the mancala has been reached and one deposited in it, sowing continues along the opponents row - this time moving right to left (sowing is done in an anticlockwise circle, so if the opponents mancala is reached, play continues on the players own row as before, until the hand is empty).
4. If the last seed of the hand ends up in the player's own mancala, they make another move.
5. If the last seed of the hand ends up in one of the player's own pits, and that pit is empty, the seeds from the opposite pit are taken and moved into the mancala, along with the seed that was sown in the empty pit.

Task Analysis

There are several advantages offered by choosing Bantumi instead of Bao as the mancala variant I base my game on:

- Slightly simpler rules enable new players to learn the game more quickly
- Smaller board and fewer seeds mean less LEDs are required on the board;
  - Makes production cheaper and quicker than with more LEDs
  - Allows the board to be smaller and therefore more portable
  - Allows the possibility of reverting to decimal output instead of binary and therefore making it easier to play
- Shorter typical game times (depending on number of starting seeds and the amount of pits) make the game more practical for travel use, as they can be over in a matter of minutes as opposed to hours for Bao.

Typically, the setup for a Bantumi game is six pits per player, with four seeds in each pit at the start of the game. This is a total of 2 x 6 x 4 = 48 seeds in play. Without doing a complex analysis of the game, it is fair to say then that each pit in iBantumi should be able to hold 48 seeds (i.e. 48 LEDs per pit). In reality, 48 will never be reached, but the value can get quite high and so it makes sense to provision for this.

Assuming 48 LEDs per pit, and 12 pits + 2 mancala, this is 14 x 48 = 672 LEDs!

If I were to modify the rules so that the game starts with two seeds per pit, this is 2 x 6 x 2 = 24 seeds in play. This is 14 x 24 = 336 LEDs!

While working on ideas for a hybrid game of Bao and Bantumi (Bao rules but with only one row per player), I came up with an idea for controlling 512 LEDs (the number required by this hybrid game) in a similar fashion to the way the DMX stage lighting control protocol works. Groups of 32 LEDs would be controlled by single PICmicros (40 pin devices, having 4 ports available - 8 LEDs per port). This would require a total of 16 PICmicros (assuming I don't use some form of array addressing - which is entirely possible).

Not only does changing the number of starting seeds in each pit alter the number of LEDs required, but it also changes the typical game length - as the number of starting seeds per pit increases, so does the average game time.

A bulleted list of questions follows overleaf.

Task Analysis

- How many pits will each player have?
  - The standard game consists of six pits per player, and I see no reason to alter this as it is tried and tested, resulting in good game play.
- How many seeds will each pit start with?
  - The choice is between two and four seeds per pit. The benefits and drawbacks of the two are summarised as follows:
    - Two starting seeds require 336 LEDs
    - With the 24 LEDs per pit required if using two starting seeds, the most obvious arrangement is a 4 by 6 array
    - Two starting seeds results in shorter average play times
    - Four starting seeds require 672 LEDs
    - With the 48 LEDs per pit required if using four starting seeds, the most obvious arrangement is an 8 by 6 array
    - Four starting seeds results in longer average play times
  - Whichever option I choose, I will still be doing the same development work; the only difference being the quantity of production work (i.e. soldering up more LEDs and PICmicro controllers). Therefore, because this project is a prototype it makes sense to reduce the production time slightly in order to allow more development time. If, when the prototype is complete, it transpires that having four starting seeds per pit results in better gameplay, it wouldn't be that much more work to alter the design to provide this, as my expectations are that the majority of the work will be programming. For these reasons, there will be two starting seeds per pit in this prototype.
- What will the input hardware consist of?
  - Tactile push switches of some sort will be used, one per pit, pressed by the player to indicate they want to initiate their turn by moving those seeds. This is an area I will need to do further research in, to find the switches that have the best ergonomics at a reasonable price.
- How many input switches are needed?
  - With six pits per player and two players, each pit having one switch, 12 (2 x 6) input switches will be needed.
- How will the input switches be read?
  - With only twelve switches required, it wouldn't be out of the question to allocate one pin on the main controlling PICmicro per switch and do it that way. However, I would quite like to gain some experience in keyboard array scanning techniques. For this reason the input switches will be connected in an array and scanned. With twelve switches, the optimal dimensions are 3 x 4, re-

Task Analysis

quiring a total of 7 pins - just less than one port on a PICmicro. Adding another pin (i.e one port, and a 4 x 4 array) allows for 16 switches to be scanned. This would allow me to add another four switches for auxiliary control such as saving/loading games etc., all while using only one port; this would negate the need for a separate PICmicro keyboard driver, as I thought I would need when developing my Bao board ideas (which would require 64 switches)

- What will the display hardware consist of?
  - LEDs, one per seed per pit.
  - LCD(s) for menu navigation.
- How many LEDs are required?
  - This has been determined by how many starting seeds there will be per pit, and how many pits per player. 336 LEDs will be required.
- How will the LEDs be controlled?
  - Several options immediately come to mind, with varying feasibility: array addressing, regional PICmicro registers, and a hybrid of the two where registers are used, and in turn drive the LEDs by array addressing. I will discuss and explain these options in more detail in the system ideas and development sections, as I am uncertain which is most appropriate. I can rule out pure array addressing however, because with 336 LEDs, it would be impossible to provide them with sufficient current for visibility.
- Will there be the facility to save games and resume at a later date?
  - This would be a very useful feature and shouldn't be too difficult to implement. Getting the core gameplay implemented has to take priority over this and so I will mark this point as optional, and consider it a bonus if I have time to implement it.
- Will the device only be for two human players, or will I develop some form of AI so that people can play against the computer?
  - As with the save and resume feature, core gamplay has to take priority over this. However, developing an AI 'player' is likely to be the most complex development task, should I have time at the end to try doing so. I will consider it a bonus if I have time to implement this.
- Will the AI have variable difficulty settings, or be 'one size fits all'?
  - As above, it will be a bonus if I have time to implement this, and I will consider it a bonus if I have time to do so. Whether the difficulty of the AI is variable depends on how the AI works and so I cannot answer this question at this stage.
- What, if any, additional outputs will there be?
  - There will be one/two 2x16 LCD screens, either one that is shared between both players (which would be difficult to position on

Task Analysis

the case such that it can be easily seen by both players) or two, one per player. This will be determined when I finalise the case design.
    - I may also add a piezo sounder, allowing audible feedback during gameplay and menu navigation.
- How, aside from the switches on each pit, will the players control the device (e.g. to start a new game/save a game)?
    - There will be a sliding on/off switch on the side of the board. If I do implement the save/resume feature, I will endeavour to make it such that when the board is switched off, the game is saved and restored automatically when the board is next switched back on. Two ways of doing this come to mind.
    - The first option would be by saving the whole game state (seed locations and which player's turn it is) to non-volatile memory every time it changes (i.e every move) and then checking if a saved game exists at power up.
    - The second option would be to connect the on/off switch to the main controlling PICmicro, and have the on/off functionality done in software, so that when the switch is in the off position, the game state is saved to non-volatile memory and then the controlling PICmicro and LED controlling PICmicros are put into a power saving sleep mode. An interrupt would be set on the controlling PICmicro to bring it out of sleep when the power switch is moved to the on position, and it would load the saved game (if present) and wake the LED controlling PICmicros. The feasibility of this solution depends on how the sleep functionality of PICmicros works, and whether it can used in the way I have described. This is an area I will need to research.
    - Navigation of the LCD menus will be achieved using four push button switches; two for scrolling through options, one for confirmation of selection, and one for cancellation of selection.
- Does the device need to be portable?
    - Yes, the device needs to portable as it is intended as a travel game. The dimensions of the Bao board I bought in Malawi seem to be at the limits of what is portable, so iBantumi should be smaller than these:
    - Folded: 345mm (length) x 100mm (width) x 35mm (height)
    - Unfolded: 345mm (length) x 200mm (width) x 17.5mm (height)
    - The smaller the better, and this is a specification point I consider very important.
- How will the device be powered?
    - In order to be portable the device should be capable of running on battery power. The minimum voltage required by PICmicros is 5V, so in order to provide for this the device will be powered by four AA batteries in series, totalling 6V. I may or may not build a power jack into the case to enable running the device from a mains power supply; again this is something I will focus on only once

Task Analysis

I've finished the core gameplay. In order to protect the electronics from overvoltage/incorrect polarity, the electronics will be connected to the power supply by means of a 7805 regulator and bridge rectifier.

- What criteria should the case design conform to?
  - Portable
  - Small
  - Lightweight
- What materials will the case be made from?
  - The case will be made from whatever materials best enable me to meet the criteria set out previously. This is an area I will need to research when designing the case.
- What microcontrollers and programming languages will be used?
  - Due to the complexity of the project, and the high IO and performance requirements, I will not be using the PICAXE system that is often used in education. Instead, I will be using blank PICmicros and programming with a combination of PIC Assembler and C. The reason for this combination is that while PIC Assembler is appropriate for the LED controlling PICmicros (as this will be a relatively simple program, whether I choose the register only or hybrid register & array option), and will enable me to ensure the timing is accurate, programming the game logic in PIC Assembler would be silly. The game logic will be more complicated than the LED control, particularly if I start working on things like an AI player. Therefore a higher level language is needed, and C seems to be the common choice with PICmicros.
  - I have no experience with either of these languages, so will need to do a lot of research and learning. I have recently read 'PIC in Practice' (D.W. Smith 2002) in preparation for this project, as I anticipated the use of PIC Assembler.

System Ideas

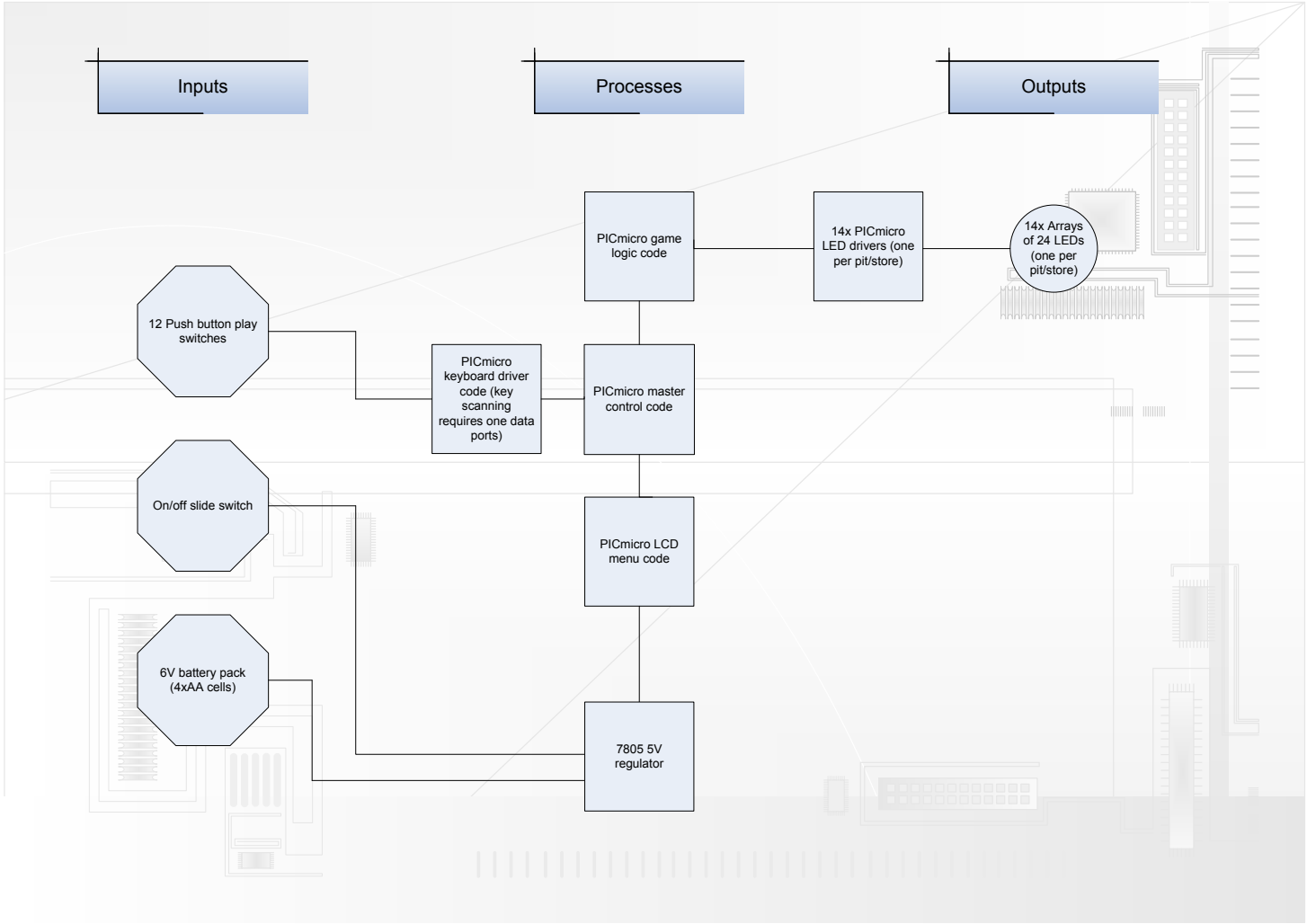


Figure 5.0 - System Idea One

System Idea One

This it the simplest of my three system ideas, consisting of only the components essential to gameplay.

To clarify the output stage:

When the PICmicro game logic code needs to update the LEDs, it calls a function of the master control code. This function outputs a serially transmitted data packet to all of the fourteen PICmicro LED drivers (which are all connected to the same pin on the main PICmicro).

Each of the 14 LED controlling PICmicros uses only the part of the packet containing information for the LEDs it controls.

System Ideas

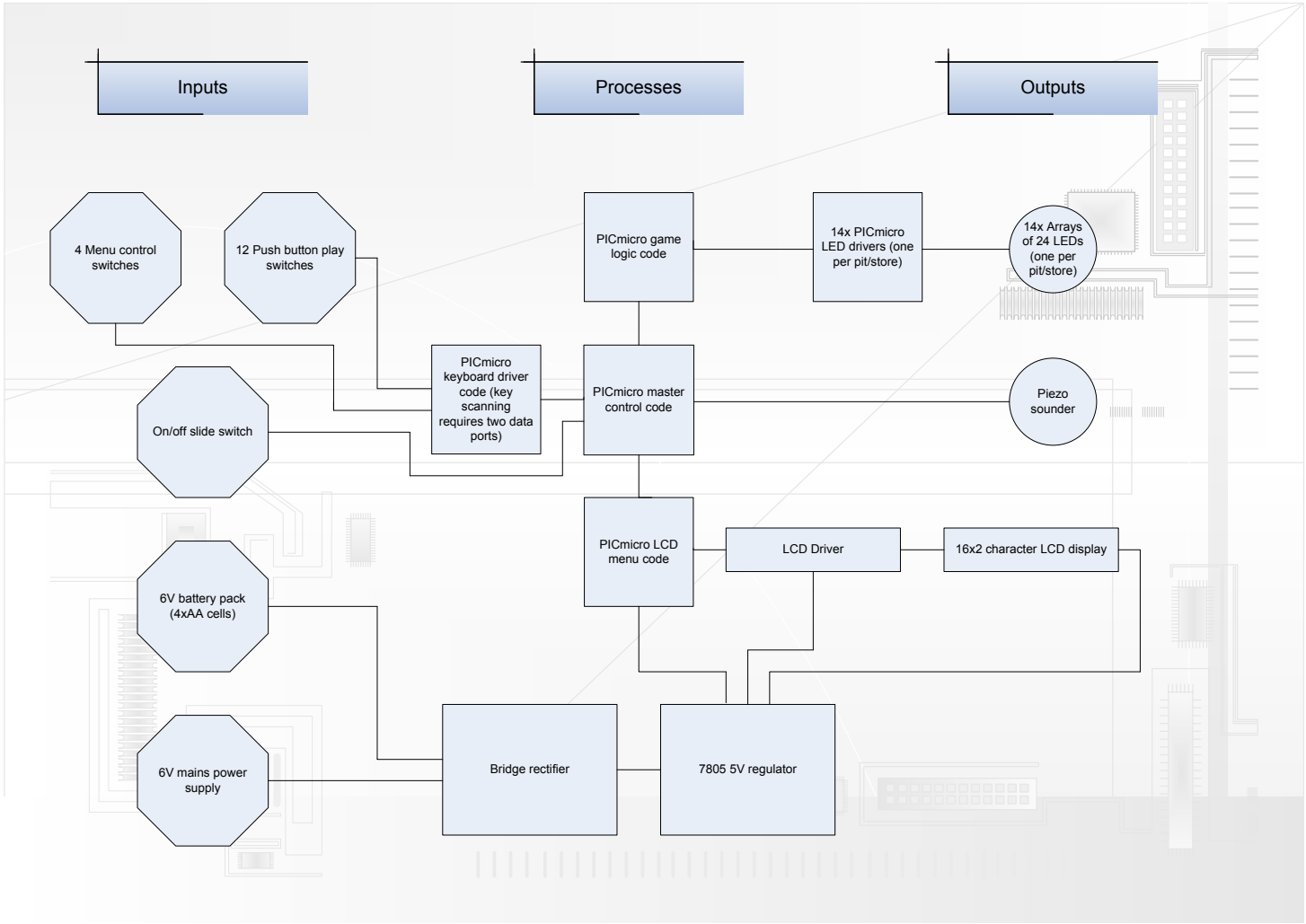


Figure 5.1 - System Idea Two

System Idea Two

System idea two adds a significant number of additional components over idea one:

- Power socket for a mains supply
- Bridge rectifier to prevent damage due to incorrectly connected power supply
- LCD driver and 16x2 character LCD display
- Four menu control buttons
- Software based power management
- Piezo sounder for audible feedback during play

System Ideas

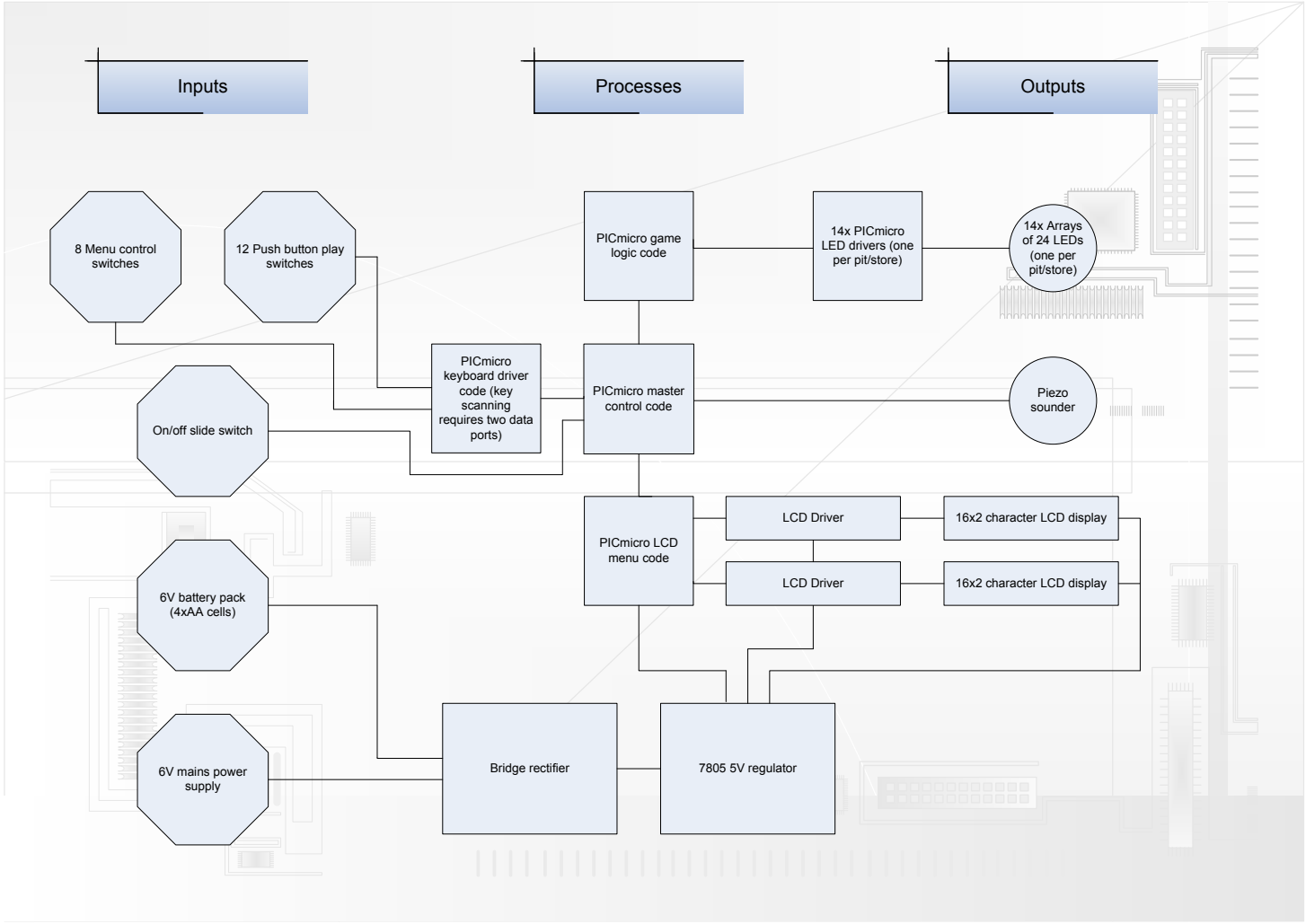


Figure 5.2 - System Idea Three

System Idea Three

System idea three is identical to system idea two in all regards except that there are two LCD drivers, two LCDs, and eight menu control buttons (four per player). This would allow me to provide independent menu functionality for each player.

System Ideas - LED Control - Arrays vs. Direct Drive

As previously mentioned, I had also recognised the option of using array addressing to control the LEDs. This is best illustrated by the schematic that is figure 6.0.

Array addressing works by connecting each row and column to a pin on the controlling PIC, configured as output.

For example, to switch LED D8 on, the pin connected to column SW6 would be switched high, and the pin connected to row SW2 switched low. All other rows and columns are disconnected.

The first reason for not using this technique is that if I was to use array addressing, each LED could only be on for 1/24 of a second; this would result in the LED running at only 1/24 of its usual brightness, which is clearly not satisfactory. The way around this would be to use set/reset latches for each LED, but this would be complicating matters even further.

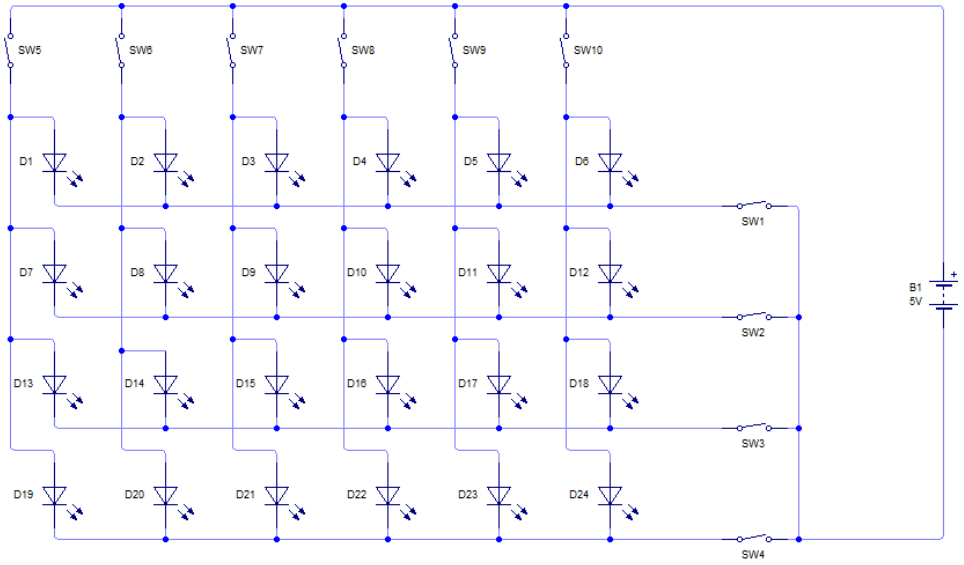


Figure 6.0 - Array addressing to control LEDs of one pit

The second reason is that it would further complicate the programming for register PICmicros, and bearing in mind that I will be programming in assembly code, it makes sense to try and keep the program as simple as possible.

Finally, using array addressing would likely result in more complicated PCB artworks than if I use direct control, and in turn would cause the PCB to be larger; not good when I have decided that a key specification point is that the device will be as small as possible given that it's a prototype.

Fortunately, I will be using array scanning to read the switches, so I will gain some experience with the array technique in doing that.

System Ideas - LCD Menus

The purpose of the LCDs is twofold.

Firstly, to provide a visual menu system to aid players in controlling the extra features, such as the save/resume functionality, sound options, and AI player. As said though, whether I implement these depends on if I have time. The second purpose is to provide in-game statistics, such as the total number of seeds on a players side and which players turn it is. If I have time to implement an AI player, I may also add other statistics (that would be calculated for the AI player anyway).

The reason for this page is to discuss whether one or two LCDs are necessary, as this is the final point I need to decide on before I can create the specification. The benefits of using two LCDs can be summarised as follows:

- Allows different information to be given to each player and therefore their own personal menu screen and statistics.
- Slightly easier to see due to better viewing angle.
- The case idea supporting two LCDs is better aesthetically than the idea supporting only one (in my opinion).

While the drawbacks can be summarised as follows:

- Adds to the total cost of an already expensive product.
- Adds to the complexity of the program.
- Adds to the complexity of the circuit (an additional four switches would be required if I used two LCDs, increasing the number of pins required for switch scanning from eight to nine).
- Adds to the complexity of the case design.

For these reasons I have decided to use only one LCD. Therefore I will be using case idea six for my case design (see overleaf - the decision was down to idea five or six, depending on whether I chose one or two LCDs), and system idea two for the schematic.

Case Ideas

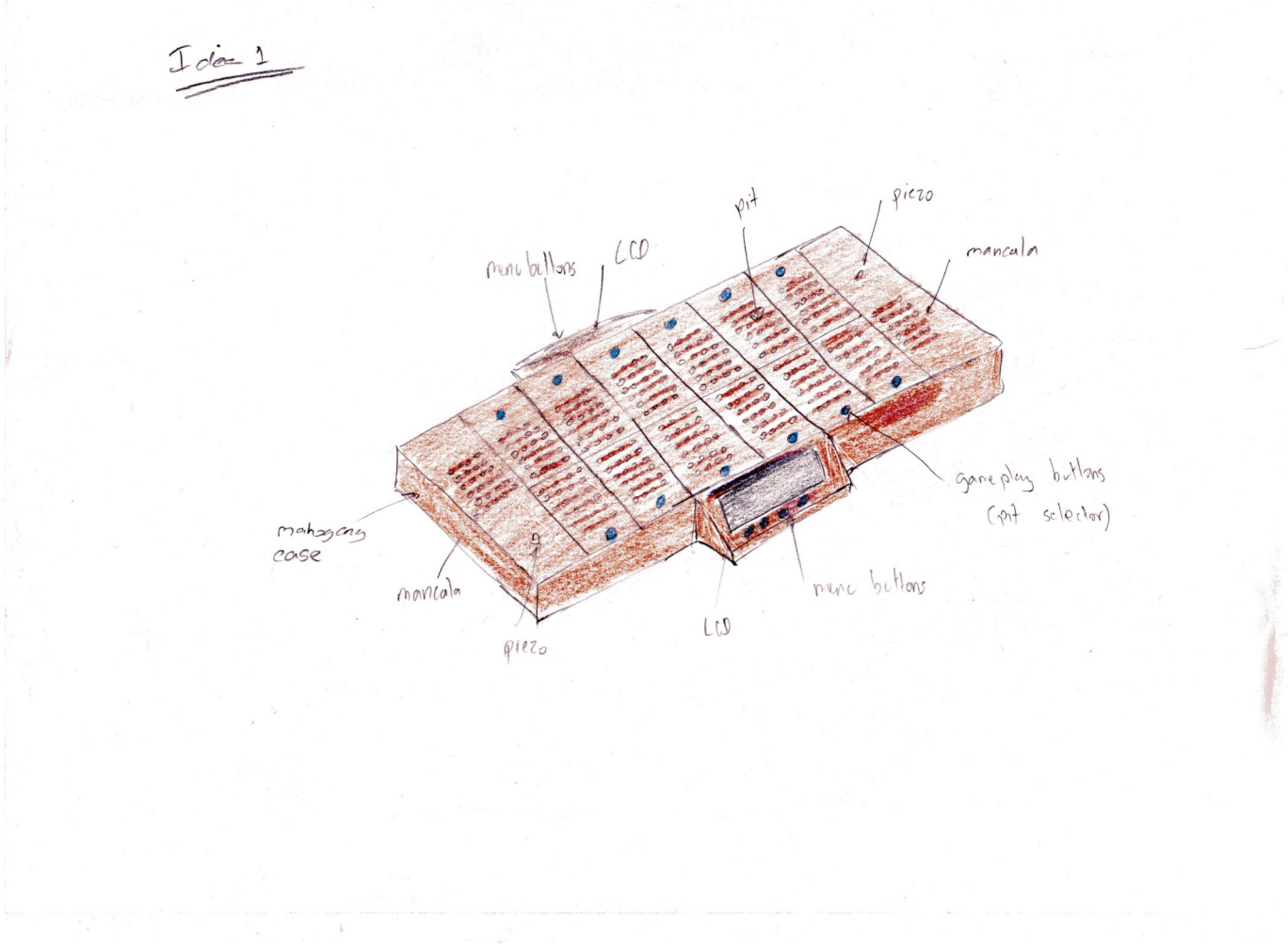


Figure 7.0 - Case Idea One

Case Idea One

In creating my case ideas, I was already limited to some extent by the requirements of the game; that is, it must be a 2 x 6 array of LEDs, with 24 LEDs in each array, and an additional array at each end acting as the players mancala.

As I have not yet decided whether to have one or two LCD screens, my case ideas offer both options.

Figure 7.0 shows my first case idea. In this idea, I have included two LCDs, one per player. The case is made from polished wood, creating a fusion of modern electronics and traditional craftsmanship should I choose this case idea.

The LED arrays are arranged 4 x 6, so that the case is shorter and wider than if they were 6 x 4.

Case Ideas

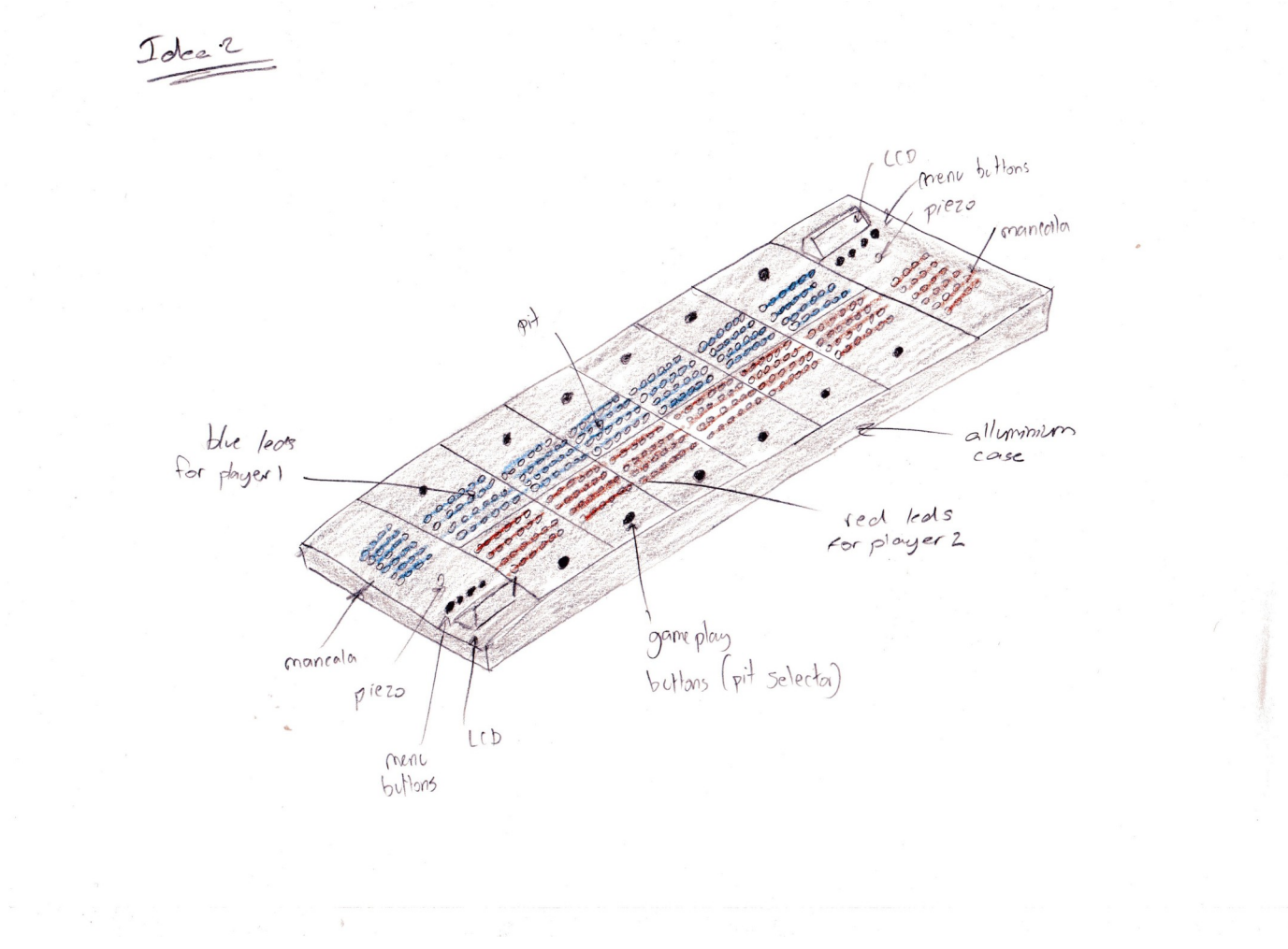


Figure 7.1 - Case Idea Two

Case Idea Two

With case idea two I wanted the shape to be noticeable different to that of idea one, while working within the constraints of requiring 336 LEDs.

In order to do this, I arranged the LED arrays 6 x 4 so that the case is longer and thinner than idea one. I moved the LCDs from the sides to the ends.

The other change in this idea is that the theme is different; instead of a wooden case construction, the case is made from aluminium.

A progression from idea one where all the LEDs were red, this idea features red LEDs for one player and blue for the other, to make it easier to differentiate between players seeds and see who is winning.

Case Ideas

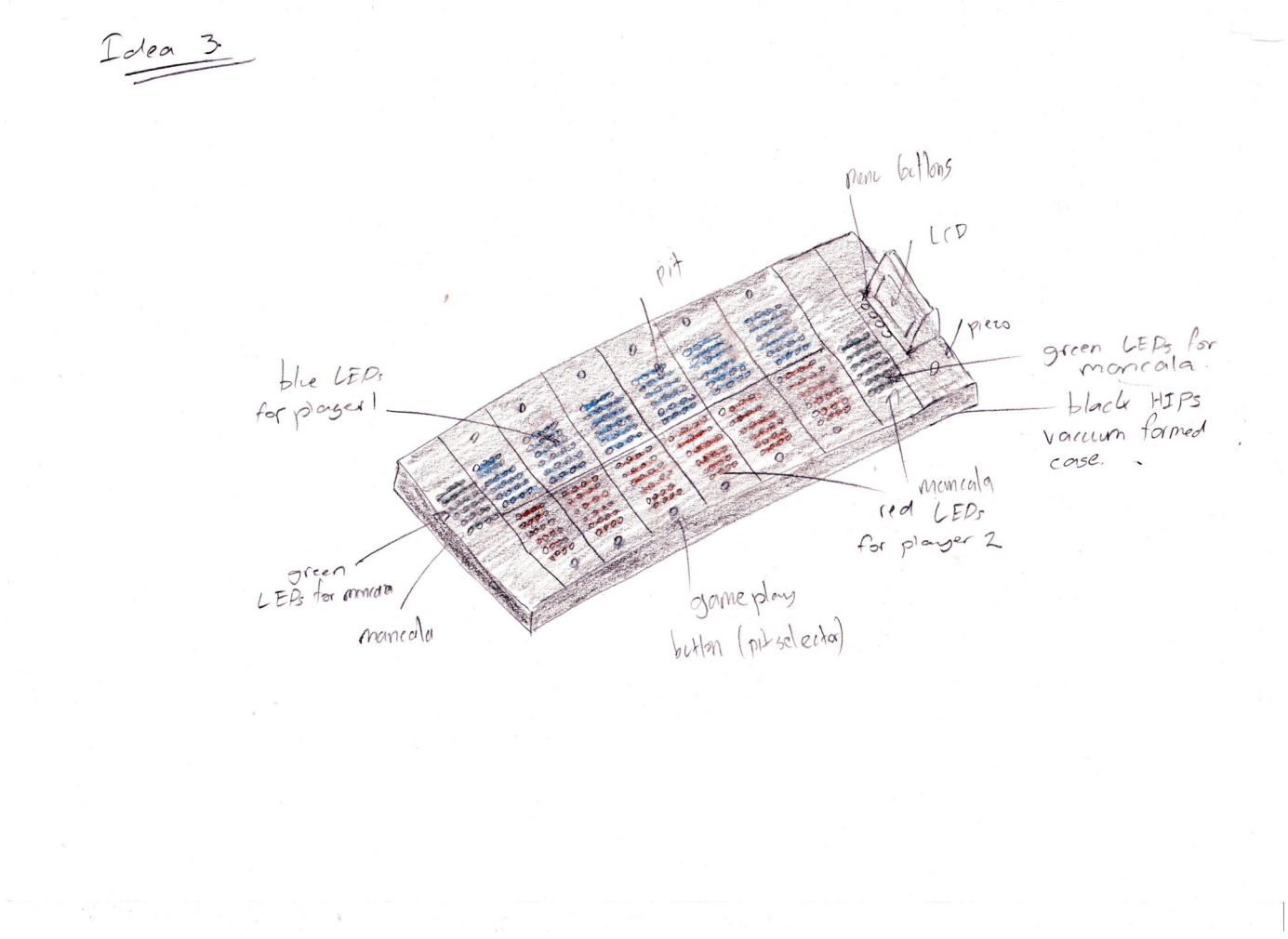


Figure 7.2 - Case Idea Three

Case Idea Three

Here is a case idea with only one LCD, positioned such that is easily visible to both players.

An improvement on idea two, the LEDs in the mancala are coloured green. There are two benefits of this:

1) New players recognise the significance of the mancala more quickly
2) Players can easily differentiate between seeds in the mancala and those still in play.

Additionally, the case is now black vacuum formed HIPS. While it may not be so aesthetically pleasing, it's important that I recognise this is a prototype, and so vac forming the case I will save valuable time on case production, allowing me to spend more time on the electronics and programming.

Case Ideas

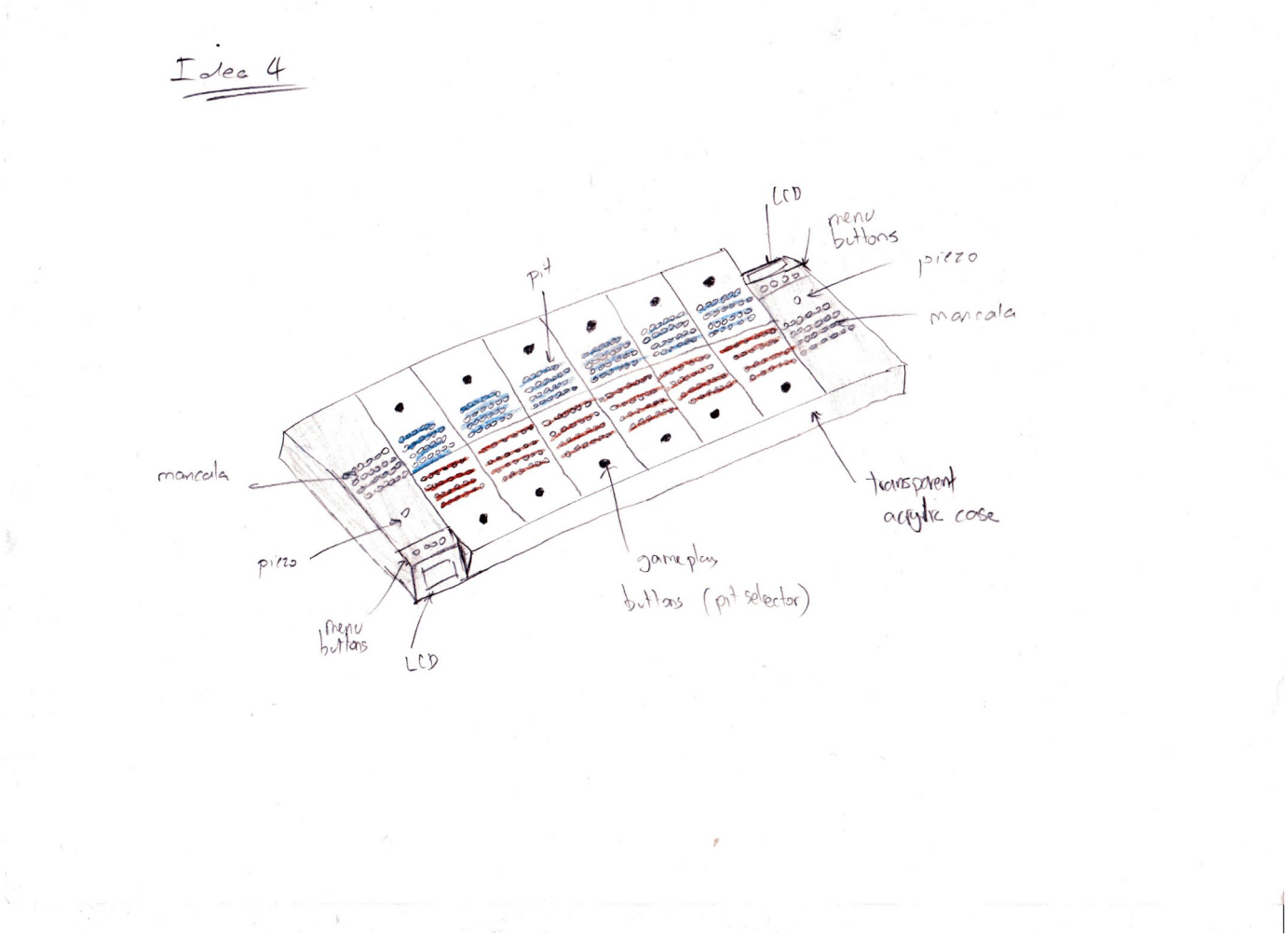


Figure 7.3 - Case Idea Four

Case Idea Four

One thing I don't like about my previous three ideas is the fact that the LCDs protrude from the main case body; in idea one they do so from the sides, and in ideas two and three from the top.

In idea four, I have positioned the LCDs such that they do not protrude from the main case body by indenting them.

I have reverted to 6 x 4 LED arrays in this idea to make better use of the extra width created in the mancala sections due to the LCDs.

The case construction is now transparent acrylic for the main body, and translucent blue acrylic for the mancala sections.

The mancala LEDs are now white instead of green, as I think will be better in terms of aesthetics.

Case Ideas

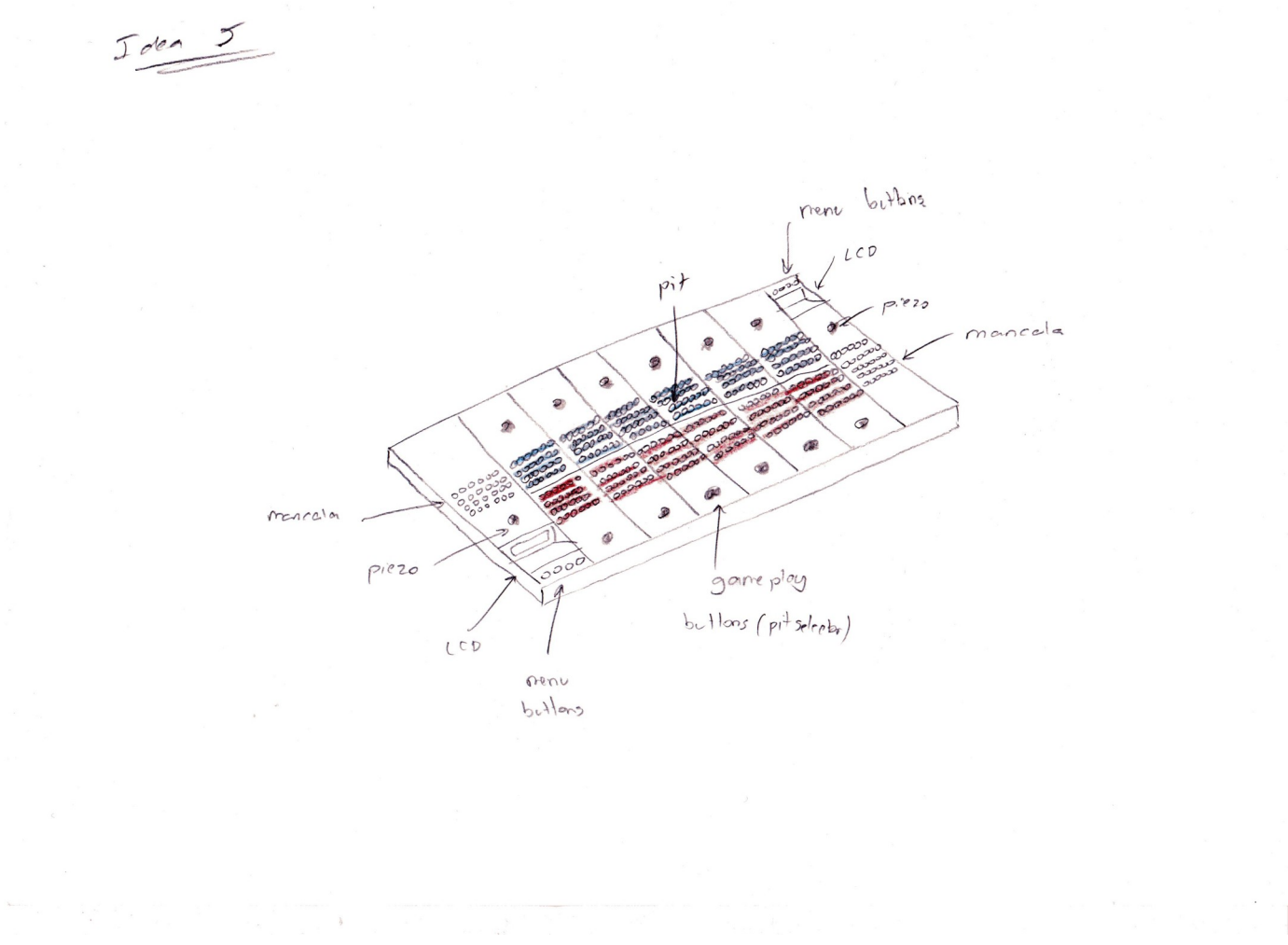


Figure 7.4 - Case Idea Five

Case Idea Five

One thing I didn't like about idea four was the fact that although the LCDs no longer protrude from the main case body, the case shape is still altered due to the angle of the screens.

To remedy this, I have positioned the LCDs such that they are in-dented in the board, as seen in figure 7.4

The other change from idea four is that the case construction is now entirely transparent acrylic.

Case Ideas

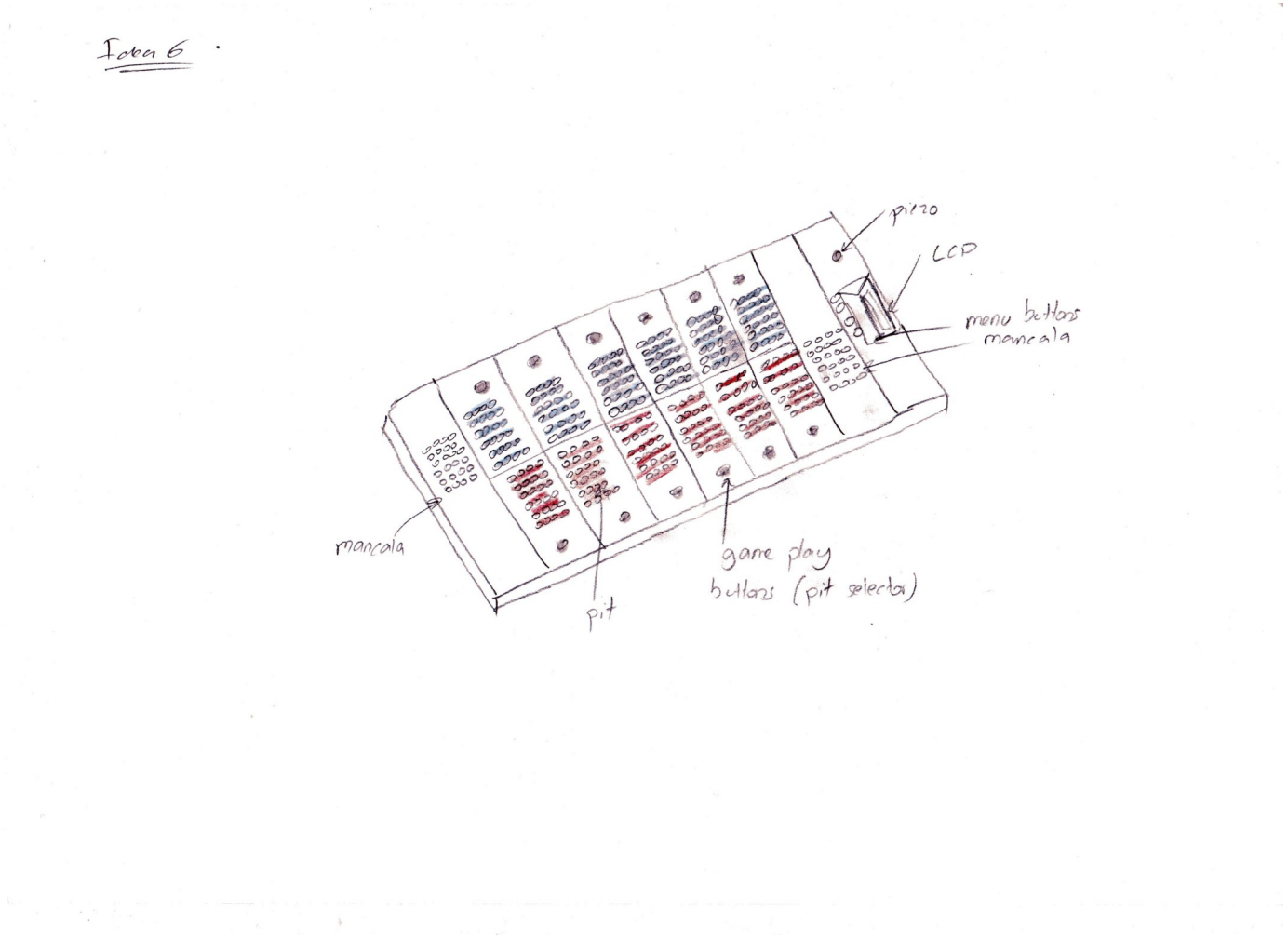


Figure 7.5 - Case Idea Six

Case Idea Six

Of the five ideas I had come up with so far, my preference is with the fifth, as it improves on all the things I didn't like about the others.

However, as already mentioned, I need to have ideas that cater for both one and two LCDs, as I still haven't decided how many to have. For this reason, I created this sixth sketch, with the only change from idea five being that there is one LCD, positioned at one end and visible to both.

As the extra room in the mancala sections is no longer required, the LED array dimensions are now 4 x 6 again. This shortens the overall board size, compensating for the extra length due to the LCD.

Specification

I have grouped the specification points based on the aspects of the design that they determine, and ranked them by number based on how important I think they are to the design.

Player Input

1. Each pit will have a push button switch, pressed by the player to initiate their move by taking the seeds from that pit into the hand.
2. There will be one LCD, shared between the two players, positioned at the end of the board such that it can be seen by both players.
3. The LCD menus will be controlled by four push button switches; two to change the selection, and two to cancel/confirm the selection.
4. This totals $12_{pit} + 4_{menu} = 16_{total}$ switches, which can be arranged and scanned in a 4x4 array, such that only 8 I/O pins (one port) are required on the main controlling PICmicro.

Case

1. The case will be based on case idea six.
2. The main material will be transparent acrylic. Some parts may be coloured translucent acrylic (e.g. red/blue for the mancala ends to indicate possession of the seeds in that area by the player).
3. The on/off switch will be a slide switch mounted on the same side of the case as the LCD.

LED Control

1. The 336 seed LEDs will be grouped into arrays of 24 LEDs, and each group controlled by a 28 pin PICmicro, programmed with assembly code, that I will refer to as LED registers.
2. Each LED will be connected directly to an output on an LED register.
3. The LED registers will be controlled by a serial pulse train from the main controlling PICmicro.
4. Therefore the LED registers will have a minimum of 25 I/O pins (24 output, 1 input for the serial pulse train).
5. The control protocol will be detailed in the development section, as and when I develop it.

Processing

1. The rules described in the following section (gameplay) will be applied by a single PICmicro, programmed using C.
2. One output pin will be required for the piezo sounder.
3. One input pin will be required for the on/off switch.

Specification

4. Eight I/O (four input, four output) will be required for the push button switches.
5. One output pin will be required for transmitting the serial pulse train to the LED registers.
6. One to eight output pins will be required for controlling the LCD, depending on how I choose to drive it (whether the driver is controlled by serial or parallel signalling).
7. Therefore a maximum of (1 + 1 + 8 + 1 + 8) = 19 I/O pins will be required on the controlling PICmicro.

Gameplay

1. The objective is to have the most seeds in your mancala at the end of play.
2. There will be six pits per player.
3. There will be one mancala per player, situated to the right of their six pits.
4. Gameplay will start with two seeds in each pit.
5. A turn consists of one or more moves.
6. Whether another move is granted depends on the outcome of the previous move.
7. A move starts when a player takes the seeds from one pit into the hand (by pressing the switch located in that pit).
8. The seeds are then sown anticlockwise one at a time around the board.
9. Seeds cannot be moved from the mancala.
10. If when a move terminates, the last seed in the hand was sown into the player's mancala, the player is granted (and is obliged to take) another move.
11. If when a move terminates, the last seed in the hand was sown into a pit directly opposing an occupied opponents pit, the seeds in that pit and the opponents pit are moved at once into the mancala of the player who initiated the move.
12. The game terminates when either player has no seeds left in play, and the winner is the player with the most seeds on their side of the board.

Extra Features

- The AI player implementation is subject to be having the time and knowledge/ability to do so.
- I may also add, accessible via the LCD menu, some "Easter eggs"; i.e. the ability to strobe all the LEDs for use as a floodlight type device.

Development Time Plan

ID	Task Name	Start	Finish	Duration
1	Christmas Holidays	15/12/2007	06/01/2008	23d
2	Spring Half Term	09/02/2008	17/02/2008	9d
3	Easter Holidays	20/03/2008	06/04/2008	18d
4	Project initialisation	01/11/2007	09/11/2007	9d
5	Core component research	11/11/2007	28/11/2007	18d
6	Familiarisation with assembly code	29/11/2007	30/11/2007	2d
7	LED register & control protocol development	03/12/2007	07/12/2007	5d
8	LED pit board schematic & artwork development	08/12/2007	09/12/2007	2d
9	Familiarisation with C & LCD research	10/12/2007	14/12/2007	5d
10	Game logic code development	14/12/2007	07/01/2008	25d
11	LCD menu development	08/01/2008	11/01/2008	4d
12	Game logic board schematic & artwork development	14/01/2008	18/01/2008	5d
13	Case dimensioning & model creation	19/01/2008	27/01/2008	9d
14	Production time plan	28/01/2008	28/01/2008	1d
15	Production	29/01/2008	12/03/2008	44d
16	Completion of documentation	13/03/2008	19/03/2008	7d

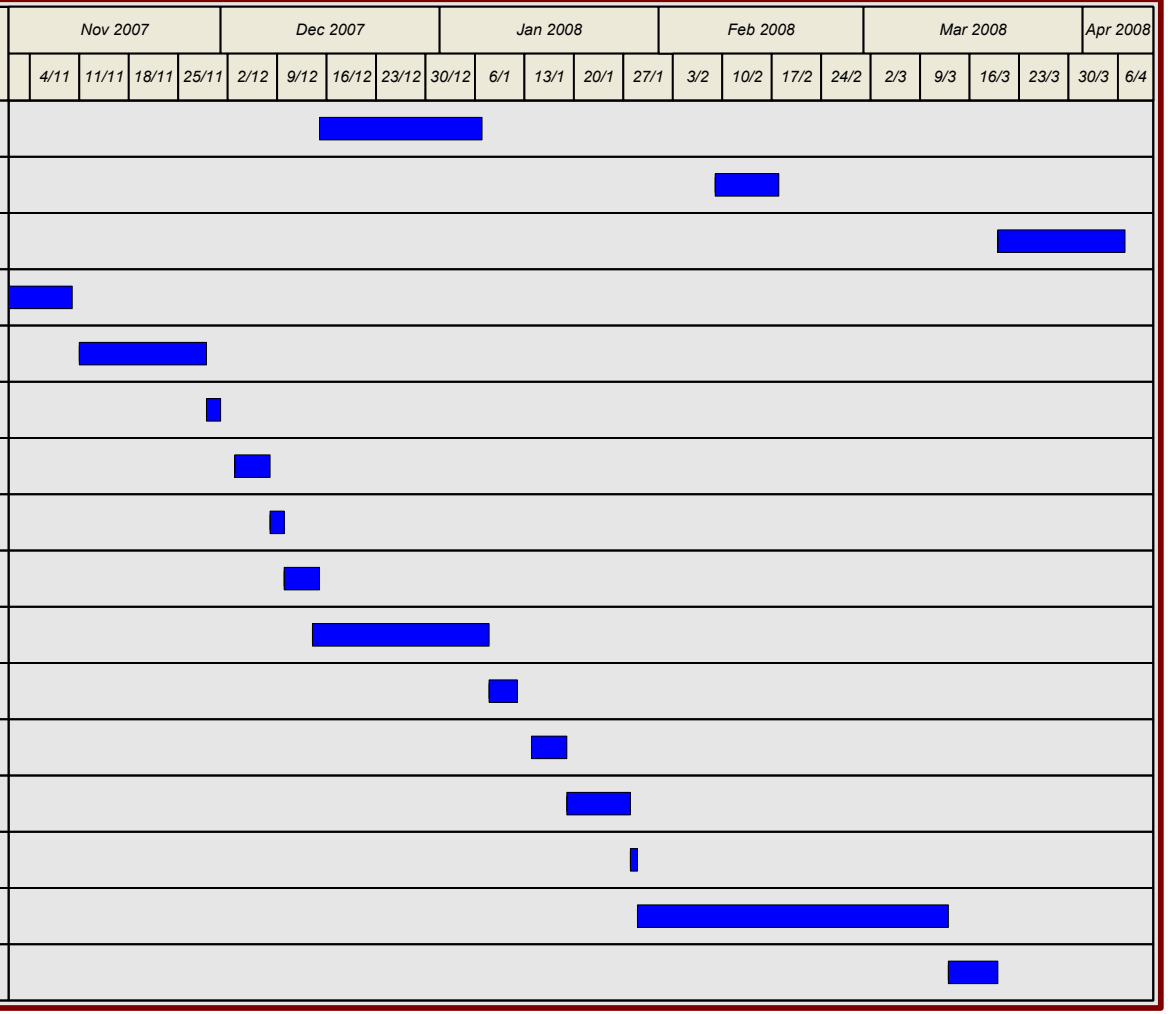


Figure 8.0 - Development time plan

Component Research - PICmicro Programmmer Research

Figure 9.0 - PICkit 2

Over the last few years the department has transitioned from using standard PICs to PICAXE, which means that there are currently no suitable programmers for PICs (PICs these days support in-circuit serial programming, in a similar fashion to PICAXE, and this is a feature I fully intend to use). As my laptop has no serial port, one of the requirements of the programmer is that it interfaces with the development machine by USB.

The most suitable programmer in the price range is the PICkit 2 Microcontroller Programmer manufactured by Microchip, and fortunately, available from Rapid Electronics for £44.

The benefits of using an in-circuit programmer and debugger can be summarised as follows:

- Allows quicker development as the PIC must not be constantly moved from the circuit to an out of circuit programmer.
- The debugging facility will aid me in development by allowing me to step through the program and inspect the values in the files of PIC.
- Reduces risk of damage to PICs due to repeated removal and reinsertion of the device into a socket.

Figure 9.0 shows the PICkit 2 programmer.

Component Research - LED Register PICmicro Research

The requirements of the LED registers can be summarised as follows:

- 28 pin package (to keep PCB size to a minimum, as even fourteen 28 pin packages will take a lot of space)
- 24 outputs
- 1 input
- As cheap as possible, given the requirement for fourteen
- Programmable with assembly code to ensure accurate timing

Figure 10.0 - PIC16F882

microchip DIRECT

Order Receipt Confirmed

Order No: 105721
Order Date: 29-Nov-2007
Order Status: Completely Shipped

Payment Option: MasterCard

Billing Address: David Piggott
n/a
Old Fox Brewery
Mill Lane
Wetley Rocks
Stoke-on-Trent
Staffordshire ST9 0BN
United Kingdom

Ship Address: David Piggott
n/a
Old Fox Brewery
Mill Lane
Wetley Rocks
Stoke-on-Trent
Staffordshire ST9 0BN
United Kingdom

Shipping Option: Seabourne Zone 1 GBP

No.	Product	Request Date	Ship Date	Unit Price (GBP)	Quantity	Total (GBP)
1.	PIC16F882-E/SP		30-Nov-2007	.94	20	18.80

Track No: 3033817
Invoice Number: 3033817

Order Status: Invoiced

Order Summary

Device Total:	18.80
Shipping:	10.55
Handling:	5.00
VAT:	6.01
Total (GBP):	40.36

Microchip VAT ID(United Kingdom): GB 505174373
This is not a tax invoice. Reverse charges may apply.

To track the progress of your shipped orders, click on the listed tracking number(s) above to be taken to the shipper's web site.

Figure 10.1 - PIC16F882

With this in mind, I went to the Microchip website and used their parametric search for PICs with the following options (see overleaf for the results):

- 28 pin device
- 25 I/O
- SPDIP package

I sorted the results by price and found the cheapest PICmicro to meet the criteria is the PIC16F882, Microchip part number PIC16F882-E/SP, at £0.94 each. However, searches on Rapid, Maplin and Radio Spares yielded no results so I was left only with the option of ordering direct from Microchip.

As a precaution against losses due to short circuit/mishandling, I've ordered excess (twenty). Because orders placed direct with Microchip ship from Thailand, I also ordered a single PIC18F252-I/SP from Rapid electronics for £4.00, along with the PICkit 2 programmer so that I can start developing the program while waiting for the PIC16F882s to arrive; if I had ordered twenty of the PIC18F252-I/SPs from Rapid, this would have cost £80.00. Figure 10.1 shows the cost breakdown for this order.

Selection Criteria

Status	=	In Production
Architecture	=	8
Memory Type	=	Flash
I/O Pins	=	25
Pin count	=	28
Internal Oscillator	=	8 MHz or 8 MHz, 32 kHz
Package	=	SPDIP

23 of 325 FLASH Devices are listed.
Parametric Search Clear search filters

Switch Views: Summary / Show All Specs

How to Use This Comparison Utility

Click the following images:
To Sort in ascending order
To Sort in descending order
To Collapse Column

Change Product Groups

FLASH - Change

To look at a different product group, select a group from the dropdown and click on "Change".

Filter	Product Family	Status	Datasheet	Volume Pricing	Architecture	Memory Type	Program Memory KBytes	Program Memory KWords	Self-write	EEPROM Data Memory	RAM	I/O Pins	Pin count	Max. CPU Speed MHz	CPU Speed MIPS	Internal Oscillator	# of A/D Ch.	Digital Communication	Timers
	PIC16F882 Ordering Options	In Production		$1.20	8	Flash	3	2	Yes	128	128	25	28	20	5	8 MHz, 32 kHz	11	1 -A/E/USART 1 -MSSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC16F883 Ordering Options	In Production		$1.43	8	Flash	7	4	Yes	256	256	25	28	20	5	8 MHz, 32 kHz	11	1 -A/E/USART 1 -MSSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC16F886 Ordering Options	In Production		$1.56	8	Flash	14	8	Yes	256	368	25	28	20	5	8 MHz, 32 kHz	11	1 -A/E/USART 1 -MSSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC16F913 Ordering Options	In Production		$1.64	8	Flash	7	4	No	256	256	25	28	20	5	8 MHz, 32 kHz	5	1 -A/E/USART 1 -SSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC16F916 Ordering Options	In Production		$1.84	8	Flash	14	8	No	256	352	25	28	20	5	8 MHz, 32 kHz	5	1 -A/E/USART 1 -SSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC18F2221 Ordering Options	In Production		$2.02	8	Flash	4	2	Yes	256	512	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2321 Ordering Options	In Production		$2.14	8	Flash	8	4	Yes	256	512	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2420 Ordering Options	In Production		$2.61	8	Flash	16	8	Yes	256	768	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC16F737 Ordering Options	In Production		$2.71	8	Flash	7	4	No	0	368	25	28	20	5	8 MHz	11	1 -A/E/USART 1 -MSSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC18F2510 Ordering Options	In Production		$2.83	8	Flash	32	16	No	0	1536	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
Filter	Product Family	Status	Datasheet	Volume Pricing	Architecture	Memory Type	Program Memory KBytes	Program Memory KWords	Self-write	EEPROM Data Memory	RAM	I/O Pins	Pin count	Max. CPU Speed MHz	CPU Speed MIPS	Internal Oscillator	# of A/D Ch.	Digital Communication	Timers
	PIC18F2423 Ordering Options	In Production		$2.89	8	Flash	16	8	Yes	256	768	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2520 Ordering Options	In Production		$3.03	8	Flash	32	16	Yes	256	1536	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC16F767 Ordering Options	In Production		$3.16	8	Flash	14	8	No	0	368	25	28	20	5	8 MHz	11	1 -A/E/USART 1 -MSSP(SPI/I2C)	2 - 8-bit 1 - 16-bi
	PIC18F2515 Ordering Options	In Production		$3.25	8	Flash	48	24	No	0	3968	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2523	In														8 MHz, 32		1 -A/E/USART	1 - 8-bit

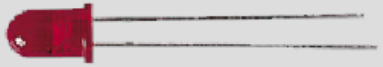
	Ordering Options >	Production		$3.30	8	Flash	32	16	Yes	256	1536	25	28	40	10	kHz	10	1 -MSSP(SPI/I2C)	3 - 16-bi
	PIC18F2525 Ordering Options >	In Production		$3.45	8	Flash	48	24	Yes	1024	3968	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2220 Ordering Options >	In Production		$3.53	8	Flash	4	2	Yes	256	512	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2320 Ordering Options >	In Production		$3.82	8	Flash	8	4	Yes	256	512	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2620 Ordering Options >	In Production		$3.87	8	Flash	64	32	Yes	1024	3968	25	28	40	10	8 MHz, 32 kHz	10	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2480 Ordering Options >	In Production		$4.30	8	Flash	16	8	Yes	256	768	25	28	40	10	8 MHz, 32 kHz	8	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
Filter	Product Family	Status	Datasheet	Volume Pricing	Architecture	Memory Type	Program Memory KBytes	Program Memory KWords	Self-write	EEPROM Data Memory	RAM	I/O Pins	Pin count	Max. CPU Speed MHz	CPU Speed MIPS	Internal Oscillator	# of A/D Ch.	Digital Communication	Timers
	PIC18F2580 Ordering Options >	In Production		$4.72	8	Flash	32	16	Yes	256	1536	25	28	40	10	8 MHz, 32 kHz	8	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2682 Ordering Options >	In Production		$5.26	8	Flash	80	40	Yes	1024	3328	25	28	40	10	8 MHz, 32 kHz	8	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi
	PIC18F2685 Ordering Options >	In Production		$5.62	8	Flash	96	48	Yes	1024	3328	25	28	40	10	8 MHz, 32 kHz	8	1 -A/E/USART 1 -MSSP(SPI/I2C)	1 - 8-bit 3 - 16-bi

Volume pricing is for budgetary use only, shown in United States dollars. The prices are representative and do not reflect final pricing. Contact your local Microchip sales representative or distributor for volume and / or discount pricing.

Component Research - LEDs

5 and 12V 5mm LED

A range of miniature 5mm LEDs incorporating an in-built series resistor enabling the LED to be directly connected to 5 or 12V supply lines.



- All devices are intensity and colour matched and housed in diffused coloured packages
- Cathode identified by flat on body
- Package style 8
- Kingbright L-7113V series (formerly L-53V)

Colour	IF	VF	VR	PTOT	Lum. int. (mcd)	View	Wave
	max.	max.	max.	max.	@IF (10mA)	angle	length
5V HE Red	17.5mA	6V	5V	85mW	30	30°	625nm
5V Green	17.5mA	6V	5V	85mW	20	30°	568nm
5V Yellow	17.5mA	6V	5V	85mW	20	30°	588nm
12V HE Red	11.5mA	14V	5V	120mW	30	30°	625nm
12V Green	11.5mA	14V	5V	120mW	20	30°	568nm
12V Yellow	11.5mA	14V	5V	120mW	20	30°	588nm

Analysis

Benefits

- 5mm diameter
- Matched intensity
- Matched viewing angles (all 30°)
- Direct connection possible without series resistor

Drawbacks

- Only available in red
- Therefore the intensity will not actually be matched as I would be mixing these LEDs with others in order to have blue and white colours too

Component Research - LEDs

Analysis

I have only included technical information for the diffused lens range because this is the only range with all the colours I require.

Benefits

- 5mm diameter
- Matched intensity
- Matched viewing angles within lens categories

Drawbacks

- Only available in red and white; not blue
- Therefore the intensity will not actually be matched as I would be mixing these LEDs with others in order to have blue as well
- It is unknown to me at this at point whether I would be able to power these directly, without a series resistor

5mm Standard LEDs

A range of high efficiency 5mm LEDs that are suitable for PCB and panel mounting applications.

They are available with diffused, transparent or water clear lenses in a variety of colours.

- All devices are intensity and colour matched
- Cathode identified by flat on body
- Kingbright L-7113 series

Colour	Wave-length (nm)	Lum. int. (mcd) @ IF (10mA)	IF max. (mA)	VF typ. (V)	VF max. (V)	VR max. (V)	View angle
Bright red	660	5	25	2.25	2.5	5	30°
HE red	625	45	30	2	2.5	5	30°
Amber	625	25	30	2	2.5	5	30°
Pure orange	610	30	25	2.05	2.5	5	30°
Yellow	588	20	30	2.1	2.5	5	30°
Green	565	20	25	2.2	2.5	5	30°
Pure green	555	5	25	2.25	2.5	5	30°
White	0.31, 0.31 *	1000 **	30	3.2	4	5	30°

Component Research - LEDs

5mm High brightness LEDs

A range of high intensity LEDs in a 5mm package that features a blade cathode which acts as a heat sink to dissipate heat at high current levels.

- High luminous intensity for use in a wide range of indication and illumination applications requiring a high output
- Manufactured using the latest technologies for long life, energy efficiency and high reliability
- Suitable for PCB and panel mounting applications
- Water clear lens

Colour	Wave-length (nm)	Lum. int. (mcd) @ IF (30mA)	IF max. (mA)	VF typ. (V)	VF max. (V)	VR max. (V)	View angle
White	0.30/0.31 *	30000	50	3.6	4.2	5	15°
Red	625	20000	50	2.2	2.6	5	15°
Orange	605	20000	50	2.2	2.6	5	15°
Yellow	590	20000	50	2.2	2.6	5	15°
Green	525	30000	50	3.6	4.2	5	15°
Cyan	505	30000	50	3.6	4.2	5	15°
Blue	470	12000	50	3.6	4.2	5	15°

Analysis

Benefits

- 5mm diameter
- Matched viewing angles
- Available in red, white and blue

Drawbacks

- Blade cathode may complicate PCB artwork
- Unmatched intensities
- It is unknown to me at this at point whether I would be able to power these directly, without a series resistor

Component Research - LEDs

5mm Ultrabright LEDs

A range of high intensity, ultrabright LEDs that employ the latest technology to produce LEDs that are highly visible and therefore suitable for a wide variety of applications. They are available with diffused, transparent or water clear lenses in a variety of colours, including white and high intensity blue.

- Cathode identified by flat on body
- Particularly suitable for signalling applications and as an alternative to filament lamps

Analysis

I have only included technical information for the red, white and blue LEDs.

Benefits

- 5mm diameter
- Available in red, white and blue

Drawbacks

- Unmatched viewing angles
- Unmatched intensities
- It is unknown to me at this at point whether I would be able to power these directly, without a series resistor

Colour	Wave-length (nm)	Lum. int. (mcd) @ IF (10mA)	IF max. (mA)	VF typ. (V)	VF max. (V)	VR max. (V)	View angle
Superbright red	660	4500	30	2.2	2.5	5	20°
Superbright red	660	1000	30	1.85	2.5	5	20°
Superbright red	640	900	30	1.85	2.5	5	20°
Superbright red	640	1400	30	1.85	2.5	5	20°
Superbright red	640	1700	30	1.85	2.5	5	20°
Superbright red	640	2800	30	1.85	2.5	5	20°
Hyper red	640	3000	40	2.45	2.6	5	50°
Hyper red	640	3500	40	2.45	2.6	5	20°
Superbright red	640	4000	30	1.85	2.5	5	20°
Hyper red	640	4000	40	2.45	2.6	5	12°
Blue	470	1000	30	3.65	4.2	5	16°
Blue	470	1200	30	3.7	4.3	5	16°
Blue	470	2200	30	3.5	4	5	15°
White	0.40/0.43*	200	30	3.8	4.5	5	20°
White	0.34/0.35*	3200	30	3.5	4	5	20°

Component Research - LEDs

The board requires 144 red LEDs, 144 blue LEDs, and 48 white LEDs; blue for one player, red for the other, and white for each player's mancala. When choosing which LEDs to use, I was looking for several things:

- 5mm diameter; any smaller would lead to problems populating the PCB
- Matched intensity between colours, so that the board does not appear 'unbalanced'
- Matched viewing angles, for the same reason
- The lowest unit price, since in total 336 are required!
- The ability to run them without a series resistor between the LED and PIC, as this will greatly simplify the PCB artwork

I started my search by looking at the 5mm LEDs section on the Rapid Electronics website and summarised the key details of all the suitable LEDs found in the previous four pages. I then compiled my findings into the following table (figure 11.0). In order to standardise the specified intensities, I took the specified intensity for each device, divided it by the current it was specified for and multiplied it by 25 (this is the maximum current the outputs of the PIC16F882 can source/sink). This does however assume that intensity is directly proportional to current.

Range	*Colour*	*Lens type*	*Costs*	*Viewing angle (°)*	*Intensity at 25mA (mcd)*	*Max Current (mA)*	*Max Voltage (V)*
5 and 12V 5mm LED	Red	Diffused	£11.52 for 144	30	30*	17.5	6
5mm Standard LEDS	Bright red	Diffused	£6.48 for 144	30	15	25	2.5
5mm Standard LEDS	HE red	Diffused	£7.20 for 144	30	112.5	30	2.5
5mm Standard LEDS	White	Diffused	£12.48 for 48	30	2500	30	4
5mm High brightness LEDs	Red	Water clear	£15.84 for 144	15	25000	50	2.6
5mm High brightness LEDs	White	Water clear	£18.72 for 48	15	16666	50	4.2
5mm High brightness LEDs	Blue	Water clear	£21.60 for 144	15	10000	50	4.2
*** at 17.5mA because this is the current consumption at 5V for this LED, given that it's designed to run without a series resistor**							

Figure 11.0 - Continued overleaf

Component Research - LEDs

Range	*Colour*	*Lens type*	*Costs*	*Viewing angle (°)*	*Intensity at 25mA (mcd)*	*Max Current (mA)*	*Max Voltage (V)*
5mm Ultrabright LEDs	Superbright Red	Water clear	£20.16 for 144	20	11250	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£8.64 for 144	20	2500	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£6.48 for 144	20	2250	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£7.20 for 144	20	3500	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£7.20 for 144	20	4250	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£12.96 for 144	20	9800	30	2.5
5mm Ultrabright LEDs	Superbright Red	Water clear	£17.28 for 144	20	10000	30	2.5
5mm Ultrabright LEDs	Hyper Red	Water clear	£12.96 for 144	50	7500	40	2.45
5mm Ultrabright LEDs	Hyper Red	Water clear	£20.88 for 144	20	8750	40	2.45
5mm Ultrabright LEDs	Hyper Red	Water clear	£20.88 for 144	12	10000	40	2.45
5mm Ultrabright LEDs	Blue	Water clear	£57.60 for 144	16	3000	30	4.2
5mm Ultrabright LEDs	Blue	Water clear	£53.28 for 144	16	3000	30	4.3
5mm Ultrabright LEDs	Blue	Water clear	£40.32 for 144	15	5500	30	4.0
5mm Ultrabright LEDs	White	Water clear	£12.48 for 48	20	500	30	4.5
5mm Ultrabright LEDs	White	Water clear	£12.96 for 48	20	8000	30	4.0

Figure 11.0 - Continued

Component Research - LEDs

From the summary table I created, figure 9.0, it can be seen that the potentially most expensive colour is blue, ranging from £21.60 to £57.60 for 144. Therefore in choosing which LEDs to use it is wise to first of all choose the cheapest blue LEDs and then select red and white LEDs to match. The cheapest blue LEDs are those from the 5mm High Brightness LEDs range at £21.60 for 144. However, they have a blade cathode which could be limiting when I am designing the PCB artwork. The lens is water clear, intensity 10000mcd at 25mA, viewing angle 15**°**, maximum current 50mA and maximum voltage 4.2V. I now need to select the cheapest red and white LEDs with the same lens type, similar intensity and viewing angles. This rules out the 5 and 12v range, and the standard range. It seems that the best match for red is one of the superbright reds from the 5mm Ultrabright LEDs range, with viewing angle 20° and brightness 10000mcd. For whites, it is not so important that viewing angles and intensities match because being in the mancala, they need not be 'balanced' with those on the rest of the board. Therefore I can loosen the constraints for the sake of reducing cost. Therefore, the most appropriate white LEDs are the 8000mcd ones from the ultrabright range (only 48p more in total than those with 500mcd intensity, and 8000mcd is a better match).

Figures 11.1 and 11.2 list two options I have put together; option two consists only of those with normal cathodes as this will give me more options when creating PCB artworks. As can be seen neither is cheap! Unfortunately there isn't a lot I can do about this, other than changing the specification to avoid blue LEDs but it is important to the design they are blue, as red and blue are 'classic' colours for two team games.

5mm Ultrabright LEDs	Superbright Red	Water clear	£17.28 for 144	20	10000	30	2.5
5mm Ultrabright LEDs	White	Water clear	£12.96 for 48	20	8000	30	4.0
5mm High brightness LEDs	Blue	Water clear	£21.60 for 144	15	10000	50	4.2
Figure 11.1 - Option One (Blue LEDs have blade cathode)				Total Cost: £51.84			
5mm Ultrabright LEDs	Superbright Red	Water clear	£12.96 for 144	20	9800	30	2.5
5mm Ultrabright LEDs	White	Water clear	£12.96 for 48	20	8000	30	4.0
5mm Ultrabright LEDs	Blue	Water clear	£40.32 for 144	15	5500	30	4.0
Figure 11.2 - Option Two (normal cathodes)				Total Cost: £66.24			

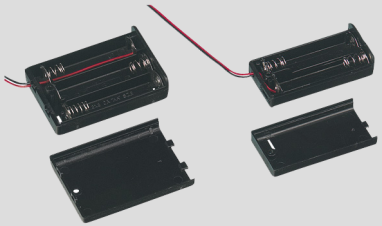
Component Research - Power Supply

The device will run from 6V power supplies; 4x AA cells as an internal power supply, and an external 6V power supply. The external power will be used in preference to the battery, which will exist to enable use as a travel game. To protect against incorrectly inserted batteries/incorrect power jack insertion, a bridge rectifier will be incorporated into the circuit. One or more 7805 5V voltage regulators will also be included, to protect against incorrect voltage as well as the protection against incorrect polarity provided by the bridge rectifier.

Battery boxes with covers

Enclosed AAA and AA size battery boxes with detachable lids, moulded in black ABS.

- Lids may be secured with the small screws provided
- Terminated in red and black stranded 26AWG wires approx. 150mm long
- Also available with fitted miniature on/off switch wired into the black (negative) wire



Price: £0.38 (for 4 X AA version, without switch)

AA Battery holders

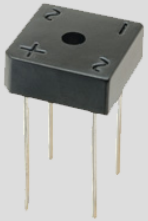
Holds batteries securely in place to provide a positive connection and maintain a low contact resistance.

- Available with PC terminals or wire leads
- Polypropylene holder material
- Nickel-plated spring steel
- Nickel-plated brass PC pins
- 26AWG tinned red and black 150mm wire leads

Price: £0.99 for 4 X AA PCB mounting version, £0.99 for 4 X AA flying leads version.

Component Research - Power Supply

8A Bridge rectifiers



8A plastic encapsulated bridge rectifiers for direct mounting to printed circuit boards.

- Dimensions 19.6(H) x 19.6(W) x 7.5(H)mm
- Pitch 13.2mm
- Weight: 6.9g
- Case to UL 94V-0

Price: £0.30 for 100V rated version

7805/7812 1A Fixed voltage regulators

A range of automatic, linear fixed voltage regulators that are designed to supply a constant regulated output voltage of either +5V DC or +12V DC. These devices operate by sensing the actual output of the voltage regulator and comparing it against an internal reference voltage. The error produced by this comparison is inverted and used to compensate the regulator output.

This feedback loop thus gives a stable output voltage that is ideal for fixed voltage use in applications such as on-card regulation for elimination of noise and distribution problems associated with single-point regulation.

- Features internal current limiting and thermal shutdown
- Delivers greater than 1A output current with adequate thermal management
- Supplied in a TO-220 package

Price: £0.25 for 5V version

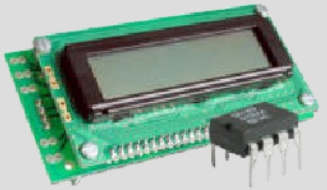
Input voltage max.		+35V
Dropout voltage		2.5V
Output voltage typ.	7805	+5V
	7812	+12V
Output current peak		1.7A
Power dissipation max.		Internally limited
Junction temperature max.		+125°C
Operating temperature range		0°C to +125°C

Component Research - LCDs

Because there are enough unknowns (in terms of time and complexity) in this project already, I have decided to try and keep the LCD as simple as possible, by using the same LCD kit as I did in my AS project (alarm clock for heavy sleepers), minus the backlight. This way I will be able to use the experience I have gained with these devices to my advantage, minimising the time required for familiarisation (some will be necessary, as the serial output method for a PIC programmed with C will be different to that with a PICAXE and BASIC).

Other options for LCDs for PICs typically involve parallel driven HD44780 or equivalent drivers, and so aswell as requiring more familiarisation time would also require more I/O on the game logic PIC.

Serial LCD



A module that allows PICAXE projects to display messages on a LCD. Interface via one single serial line or i2c bus. Optional clock upgrade (AXE034 -purchase separately) adds real time clock and alarm features to the module.

Price: £14.10

Component Research - Switches

Miniature tactile switch

12 x 12mm PCB mounted tactile switches.

- Positive switch action
- Round button
- Single pole normally open contacts
- Diptronic DTS series

Price: £0.20

According to the specification section titled 'Player input', point two, the board should have (green) illuminated buttons or normal buttons with adjacent LEDs for each pit. I have opted for the latter; although I found suitable tactile switches with built in illumination, I think that these miniature tactile switches combined with adjacent LEDs will be better in terms of aesthetics (and cheaper!)

These switches will also be ideal for the LCD menu buttons.

The adjacent LEDs (12 in total) can be connected to an extra (15th) PIC16F882 LED register.

Matching buttons for the above switches. I will be using the black square version.

Miniature tactile switch buttons

A range of square and round buttons for use with the 12 x 12mm tactile switches, (not suitable for round head switches).

- Round 13mm dia x 5.5mm
- Square 12mm square x 5.5mm

Price: £0.05

Component Research - Game Logic PICmicro

PIC18F252-I/SP

This powerful 10MIPs (100ns instruction execution) yet easy-to-program (only 77 single word instructions) CMOS FLASH-based 8-bit microcontroller packs the powerful PIC architecture into a 28/40/44-pin package (including the low footprint Skinny DIL package) and is upwardly compatible with the PIC16C5x, PIC12Cxxx, PIC16Cxx and PIC17Cxx devices. This provides a seamless migration path of software code to higher levels of hardware integration. The range of devices is ideal for manufacturing equipment, instrumentation and monitoring, data acquisition, power conditioning, environmental monitoring, telecom and consumer audio/video applications. Runs from standard 5V DC rail.

- Multi-channel ADC
- Parallel Slave Port on 252, 452, 4420 and 4520 devices
- 4 timers
- WDT with on-board RC oscillator
- 'C' compiler friendly development environment
- Single supply 5V In-Circuit Serial Programming via 2-pins
- Compatible 10-bit ADC with fast sampling rate
- Serial port either 3-wire SPI, or 2-wire I^2C bus
- Addressable USART supports RS-485 and RS-232
- High current sink/source 25mA/25mA

Price: £4.00

The criteria I set when choosing the game logic PICmicro are as follows:

- C compiler friendly instruction set
- Reasonably large program memory
- Sufficient I/O (19 according to specification) plus some spare
- In Circuit Serial Programming support

The 18F252-I/SP meets all these and is supplied in a 28 pin DIL package. It has 32k of program memory (highest available in the range but only 20p more).

Unfortunately, this means very little to me; the only perspective I have is that my AS project (heavy sleeper's alarm clock) used all 2kB of program memory and that I had to cut some features in order to fit it in and that the program for iBantumi is going to be many times more complicated.

LED Control Protocol - Ideas

As already stated, the LED registers (fourteen in number) will all be controlled by the same serial pulse train, transmitted as a single packet of data. Therefore transmission of only one packet is necessary to update all LEDs.

To simplify issues with regard to timing, there will be no clock or specified data transmission data rate, and instead the status of each bit will be given by its pulse length. This is best illustrated by figure 12.0:

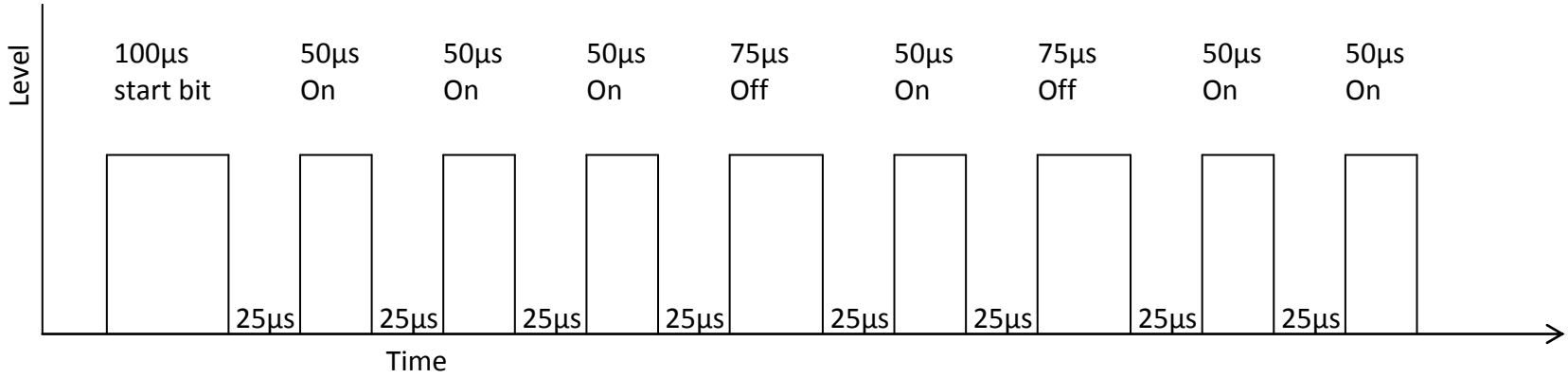


Figure 12.0 - Serial control protocol

The above example would transmit the byte '11101011', where each bit represents the status of a single LED on the board. Since the only identifying property of a bit is where in the sequence it occurs, the start pulse is necessary so that the registers know which LED it refers to.

As already specified, there will be fourteen registers, each controlling 24 LEDs (one pit). With 336 LEDs in total, each packet will consist of 337 pulses.

I chose the timings such that a minimum packet rate of 25Hz is possible; this can be seen from the fact that assuming all LEDs are off (maximum packet duration) the time period is 1(100) + 336(75) + 336(25) = 33700µs = 0.0337s. F = 1/T = 1/0.0337 = 29.7Hz.

LED Register Code Development - 0.1

I have been using MPLAB version 8.01 most of the time (prior to this it was 7.60, 7.62 and 8.00), and the PICKit 2 2.40 software and latest firmware (2.1000). When setting up the project in MPLAB I elected to use a linker script as this is the recommended option in the tutorial. The project consists of three files:

- 16f822.lkr - the linker script (included with MPLAB)
- p16F882.inc - processor specific variable definitions (included with MPLAB)
- 16F882TMP0.ASM - the actual program I am developing

Figure 13.0 shows the workspace in MPLAB I have set up for developing the register code.

- Right hand pane is the actual code
- Top left pane is project file browser
- Top centre pane is output showing build information etc.
- Middle left pane is the watch - shows the values of selected registers for debugging purposes
- Lower left pane is the stimulus workbook - I have it set up to fire asynchronous pulses of 50, 75 and 100µs to pin 3, port E (the serial input pin)

I am using the built in module, MPLAB SIM for debugging purposes.

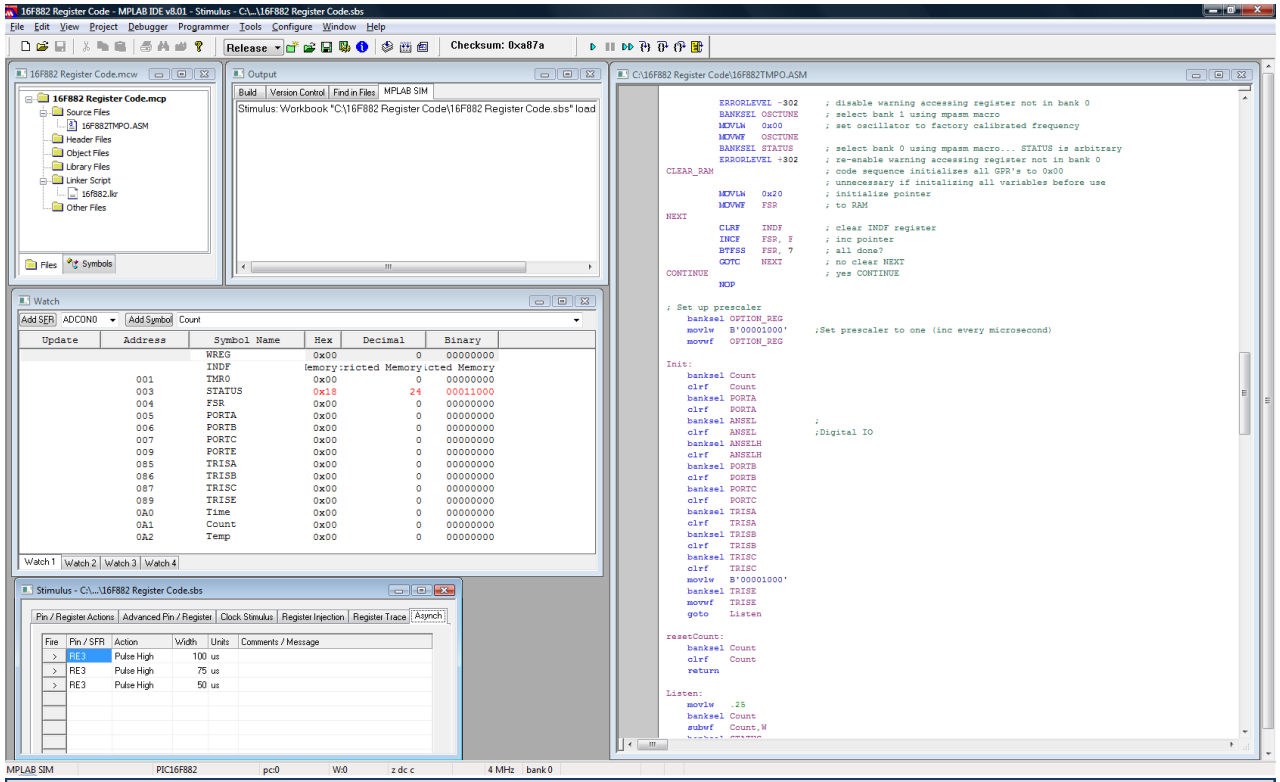


Figure 13.0 - MPLAB LED Register Project

Figure 13.1 - Register Code 0.1.

```
;*****************************************************************
;                                                                *
;   Filename:               16F882 Register Code                 *
;   File Version:           0.1                                  *
;                                                                *
;   Author:                 David Piggott                        *
;   Company:                piggott.me.uk                        *
;                                                                *
;*****************************************************************

    LIST    p=16F882         ; list directive to define processor
    #INCLUDE <p16F882.inc>       ; processor specific variable definitions

    __CONFIG _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF & _BOR_OFF &
_CPD_OFF & _CP_OFF & _MCLRE_ON & _PWRTE_ON & _WDT_OFF &
_INTRC_OSC_NOCLKOUT

    __CONFIG _CONFIG2, _WRT_OFF & _BOR40V

; '__CONFIG' directive is used to embed configuration word within .asm file.
; The lables following the directive are located in the respective .inc file.
; See data sheet for additional information on configuration word settings.
; Note that the _DEBUG_ON is changed by selecting a debugger or programmer.
; There isn't much advantage to setting it in the code.

;-----------------------------------------------------------------
;
; Variable Definitions
;
; Available Data Memory (also RAM) is divided into 4 Banks.  Special Function
; Registers, General Purpose Registers, and Access Registers are shown below:
;
;        Bank 0      Bank 1      Bank 2       Bank 3
; SFR     0x00-0x1F   0x80-0x9F   0x100-0x10F   0x180-0x18D
; GPR     0x20-0x7F   0xA0-0xBF   -             -
; ACCESS  0x70-0x7F   0xF0-0xFF   0x170-0x17F   0x1F0-0x1FF
;
; The Access Bank Registers are mutual across each address space.
;
;-----------------------------------------------------------------

; UDATA declares a section of uninitialized data
VARIABLES  UDATA         ; VARIABLES is the name of the section of memory
Time          RES    1     ; uninitialized data, placed by linker in GPR's.
Count         RES    1
Temp          RES           1

; UDATA_SHR declares a section of shared (across all banks) uninitialized data
INT_VAR    UDATA_SHR      ; INT_VAR is the section name in Access RAM
```

LED Register Code Development - 0.1

```
w_temp    RES  1     ; variable used for context saving
status_temp RES  1     ; variable used for context saving

;-----------------------------------------------------------
; EEPROM INITIALIZATION
;
; The 16F882 has 128 bytes of non-volatile EEPROM, starting at address 0x2100
;
;-----------------------------------------------------------

          ORG  0x2100
          DE   0x00, 0x01, 0x02, 0x03

;-----------------------------------------------------------
; RESET VECTOR
;-----------------------------------------------------------

RESET_VECTOR  CODE  0x0000
          goto   START        ; When using debug header, ICD2 may not stop
                              ; on instruction 0 during reset.

;-----------------------------------------------------------
; INTERRUPT SERVICE ROUTINE
;-----------------------------------------------------------

INT_VECTOR    CODE  0x0004 ; Interrupt Vector Location
          goto   INTERRUPT

PROGRAM   CODE

INTERRUPT             ; Relocatable Interrupt Service Routine

;     Context saving for ISR
          MOVWF  w_temp       ; save off current W register contents
          MOVF   STATUS,w     ; move status register into W register
          MOVWF  status_temp  ; save off contents of STATUS register

;-----------------------------------------------------------
; SAMPLE INTERRUPT SERVICE ROUTINE
;-----------------------------------------------------------

;     If the interrupt came from the timer, execute the TMR0 interrupt
;     service routine. This may be removed in addition to the sample
;     program below.
;       BTFSC  INTCON, T0IF     ; Uncomment this line to test sample code
;       CALL   TMR0_ISR         ; Uncomment this line to test sample code

;-----------------------------------------------------------
; END OF SAMPLE INTERRUPT SERVICE ROUTINE
;-----------------------------------------------------------
```

LED Register Code Development - 0.1

```
;     Restore context before returning from interrupt
      MOVF   status_temp,w    ; retrieve copy of STATUS register
      MOVWF  STATUS           ; restore pre-isr STATUS register contents
      SWAPF  w_temp,f
      SWAPF  w_temp,w         ; restore pre-isr W register contents
      RETFIE                  ; return from interrupt

;-----------------------------------------------------------
; MAIN PROGRAM
;-----------------------------------------------------------

START

;-----------------------------------------------------------
; SET OSCILLATOR TO FACTORY FREQUENCY AND CLEAR GPR's
;-----------------------------------------------------------

      ERRORLEVEL -302   ; disable warning accessing register not in bank 0
      BANKSEL OSCTUNE   ; select bank 1 using mpasm macro
      MOVLW  0x00       ; set oscillator to factory calibrated frequency
      MOVWF  OSCTUNE
      BANKSEL STATUS    ; select bank 0 using mpasm macro... STATUS is arbitrary
      ERRORLEVEL +302   ; re-enable warning accessing register not in bank 0
CLEAR_RAM               ; code sequence initializes all GPR's to 0x00
                        ; unnecessary if initallizing all variables before use
      MOVLW  0x20       ; initialize pointer
      MOVWF  FSR        ; to RAM
NEXT
      CLRF   INDF       ; clear INDF register
      INCF   FSR, F     ; inc pointer
      BTFSS  FSR, 7     ; all done?
      GOTO   NEXT       ; no clear NEXT
CONTINUE                ; yes CONTINUE
      NOP

; Set up prescaler
      banksel     OPTION_REG
      movlw       B'00001000'       ;Set prescaler to one (inc every microsecond)
      movwf       OPTION_REG

Init:
      banksel Count
      clrf    Count
      banksel     PORTA
      clrf    PORTA
      banksel     ANSEL             ;
      clrf    ANSEL                 ;Digital IO
      banksel     ANSELH
      clrf    ANSELH
      banksel     PORTB
      clrf    PORTB
```

LED Register Code Development - 0.1

```
    banksel     PORTC
    clrf    PORTC
    banksel     TRISA
    clrf    TRISA
    banksel     TRISB
    clrf    TRISB
    banksel     TRISC
    clrf    TRISC
    movlw       B'00001000'
    banksel     TRISE
    movwf       TRISE
    goto    Listen

resetCount:
    banksel     Count
    clrf    Count
    return

Listen:
    movlw       .25
    banksel     Count
    subwf   Count,W
    banksel     STATUS
    btfsc   STATUS,2
    call    resetCount
    banksel     PORTE
    btfss   PORTE,3
    goto    Listen

                        ;Prescaler is divide by one.
Timer:
    banksel     TMR0
    clrf    TMR0
Loop:
    banksel     PORTE
    btfsc   PORTE,3
    goto    Loop
    banksel     TMR0
    movfw       TMR0

Decision:
    banksel     Time
    movwf       Time        ;Subtract pulse length from 63
    sublw   .63
    banksel     STATUS
    btfsc   STATUS,0    ;If answer is +ve/0, pulse length was less than/equal to 63
    goto    SwitchOn    ;Therefore switch on

    banksel     Time
    movfw       Time        ;Subtract pulse length from 88
    sublw   .88
    banksel     STATUS
```

LED Register Code Development - 0.1

```
    btfsc  STATUS,0    ;If answer is +ve/0, pulse length was less than/equal to 88
    goto   SwitchOff   ;Therefore switch off

    movlw      B'00000001';Else, pulse was a start bit - update the count
    banksel    Count
    movwf      Count
    goto   Listen

SwitchOn:
    clrw
    banksel    Count
    iorwf Count,W
    banksel STATUS
    btfsc  STATUS,2
    goto   Listen

    call   SetPort
    banksel    FSR
    movwf      FSR
    call   Lookup
    banksel    INDF
    iorwf INDF,1
    banksel    Count
    incf   Count
    goto   Listen

SwitchOff:
    clrw
    banksel    Count
    iorwf Count,W
    banksel STATUS
    btfsc  STATUS,2
    goto   Listen

    call   SetPort
    banksel    FSR
    movwf      FSR
    call   Lookup
    movwf      Temp
    comf Temp
    movfw      Temp
    banksel    INDF
    andwfINDF,1
    banksel    Count
    incf   Count
    goto   Listen

SetPort:
    banksel    Count
    movfw      Count
    addwfPCL
```

LED Register Code Development - 0.1

```
nop
retlw PORTA;1
retlw PORTA;2
retlw PORTA;3
retlw PORTA;4
retlw PORTA;5
retlw PORTA;6
retlw PORTA;7
retlw PORTA;8
retlw PORTB;9
retlw PORTB;10
retlw PORTB;11
retlw PORTB;12
retlw PORTB;13
retlw PORTB;14
retlw PORTB;15
retlw PORTB;16
retlw PORTC;17
retlw PORTC;18
retlw PORTC;19
retlw PORTC;20
retlw PORTC;21
retlw PORTC;22
retlw PORTC;23
retlw PORTC;24

Lookup:
    banksel    Count
    movfw      Count
    addwfPCL
    nop
    retlw B'00000001';1
    retlw B'00000010';2
    retlw B'00000100';3
    retlw B'00001000';4
    retlw B'00010000';5
    retlw B'00100000';6
    retlw B'01000000';7
    retlw B'10000000';8
    retlw B'00000001';9
    retlw B'00000010';10
    retlw B'00000100';11
    retlw B'00001000';12
    retlw B'00010000';13
    retlw B'00100000';14
    retlw B'01000000';15
    retlw B'10000000';16
    retlw B'00000001';17
    retlw B'00000010';18
    retlw B'00000100';19
    retlw B'00001000';20
```

LED Register Code Development - 0.1

```
retlw B'00010000';21
retlw B'00100000';22
retlw B'01000000';23
retlw B'10000000';24

    END
```

LED Register Code Development - 0.1

It almost goes without saying that the program didn't work flawlessly (in fact at all) first time. I encountered several problems along the way and so for this reason and the amount of pages required to include this version, I haven't and will not include the program as it was at every point before testing with the MPLAB SIM. The problem which delayed me the most however was with I/O.

I had already written a test program that switches on all twenty-four outputs so I could verify my prototyping PCB was working, so I shouldn't really have had this problem but nevertheless it happened. The problem was with the init stage was as shown in figure 13.2. When I stepped through the program using MPLAB SIM and providing stimulus on PORTE,3 with the stimulus workbook, I had set up the watch (figure 13.3) with all the significant files used in the program, and strangely enough these changed (almost) exactly as expected to indicate the program was functioning correctly, with the slight problem that when it got to the output stage, PORTA,B and C simply wouldn't change.

```
Init:
        banksel Count
        clrf    Count
        banksel         PORTA
        clrf    PORTA
        banksel         PORTB
        clrf    PORTB
        banksel         PORTC
        clrf    PORTC
        banksel         TRISA
        clrf    TRISA
        banksel         TRISB
        clrf    TRISB
        banksel         TRISC
        clrf    TRISC
        movlw           B'00001000'
        banksel         TRISE
        movwf           TRISE
        goto    Listen
```

Figure 13.2 - Problematic init section

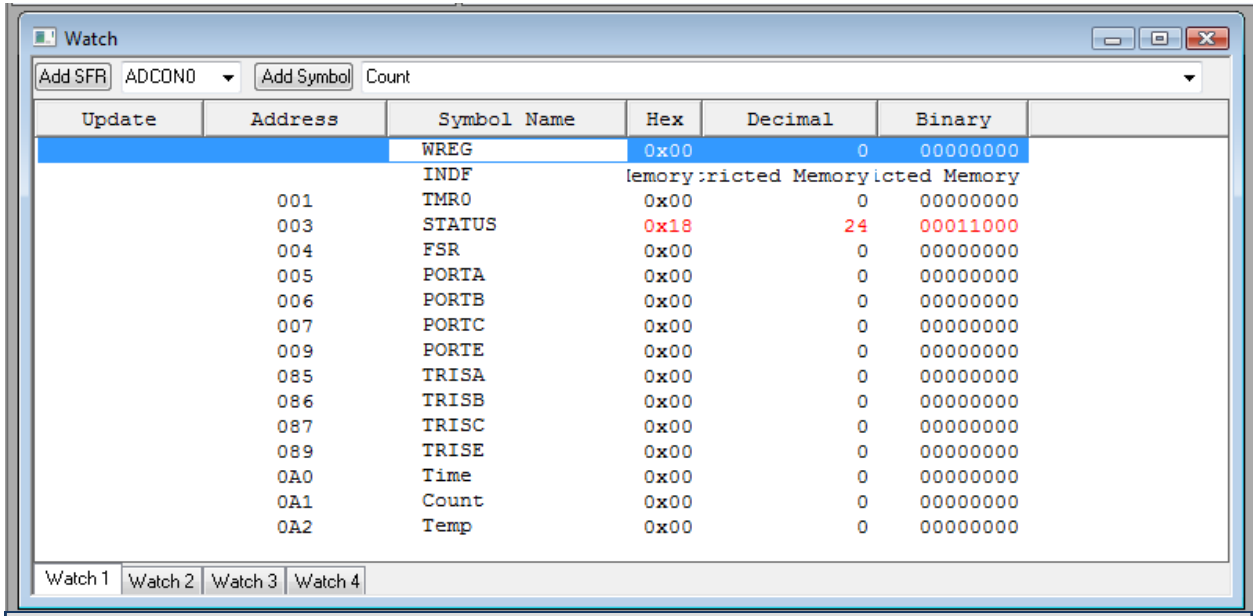


Figure 13.3 - The Watch panel

LED Register Code Development - 0.1

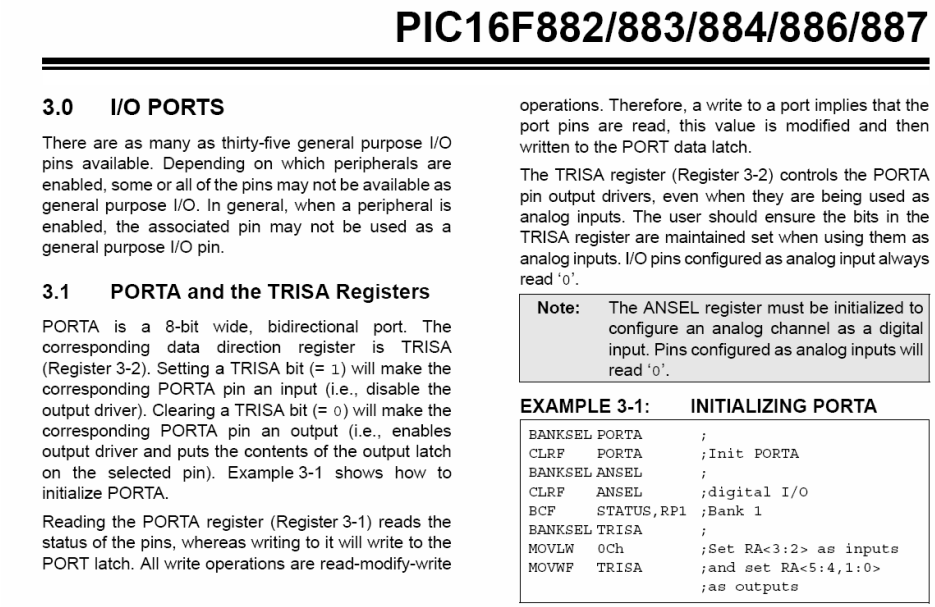
At first I thought this must be due to a bug in MPLAB (as my experience with the PICKit 2 Programmer had told me - see 'Problematic Programming'), so I checked the Microchip website and indeed there was a new version available for download so I updated from 7.62 to 8.00, and then again a few days later from 8.00 to 8.01. Neither of these updates solved the problems, and after having found neither mention of nor solution for the problem via the Google search engine, I sent a support request to Microchip via http://support.microchip.com/ on the 22nd of December at 7:14 PM. Figure 13.6 overleaf shows the MPLAB WebTicket system with the description of the problem I gave and the solution I received at on the 24th December at 10:52pm - this made a nice Christmas present when I checked my e-mail the following day!

The solution I was given shown in figure 11.4:

```
Some of these pins are shared with ADC moudle. The ANSEL register must be initialized to
configure an analog channel as a digital I/O.
e.g. ; initializing PORTA
BANKSEL PORTA ;
CLRF PORTA ;Init PORTA
BANKSEL ANSEL ;
CLRF ANSEL ;digital I/O
BCF STATUS,RP1 ;Bank 1
BANKSEL TRISA ;
MOVLW 0Ch ;Set RA<3:2> as inputs
MOVWF TRISA ;and set RA<5:4,1:0>
```

Figure 13.4 - Resolution returned

The problem was due to the fact that when writing the test program to switch all the LEDs on I had been using example code so hadn't followed it entirely. This was made worse by the fact the datasheet doesn't document this; it only says the ANSEL register must be initialised to configure an analog channel as a digital input (see figure 13.5). It doesn't mention outputs.



PIC16F882/883/884/886/887

3.0 I/O PORTS

There are as many as thirty-five general purpose I/O pins available. Depending on which peripherals are enabled, some or all of the pins may not be available as general purpose I/O. In general, when a peripheral is enabled, the associated pin may not be used as a general purpose I/O pin.

3.1 PORTA and the TRISA Registers

PORTA is a 8-bit wide, bidirectional port. The corresponding data direction register is TRISA (Register 3-2). Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., disable the output driver). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., enables output driver and puts the contents of the output latch on the selected pin). Example 3-1 shows how to initialize PORTA.

Reading the PORTA register (Register 3-1) reads the status of the pins, whereas writing to it will write to the PORT latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, this value is modified and then written to the PORT data latch.

The TRISA register (Register 3-2) controls the PORTA pin output drivers, even when they are being used as analog inputs. The user should ensure the bits in the TRISA register are maintained set when using them as analog inputs. I/O pins configured as analog input always read '0'.

Note: The ANSEL register must be initialized to configure an analog channel as a digital input. Pins configured as analog inputs will read '0'.

EXAMPLE 3-1: INITIALIZING PORTA

```
BANKSEL PORTA        ;
CLRF    PORTA        ;Init PORTA
BANKSEL ANSEL        ;
CLRF    ANSEL        ;digital I/O
BCF     STATUS,RP1   ;Bank 1
BANKSEL TRISA        ;
MOVLW   0Ch          ;Set RA<3:2> as inputs
MOVWF   TRISA        ;and set RA<5:4,1:0>
                     ;as outputs
```

Figure 13.5 - PIC16F822 Datasheet I/O Ports section

LED Register Code Development - 0.1

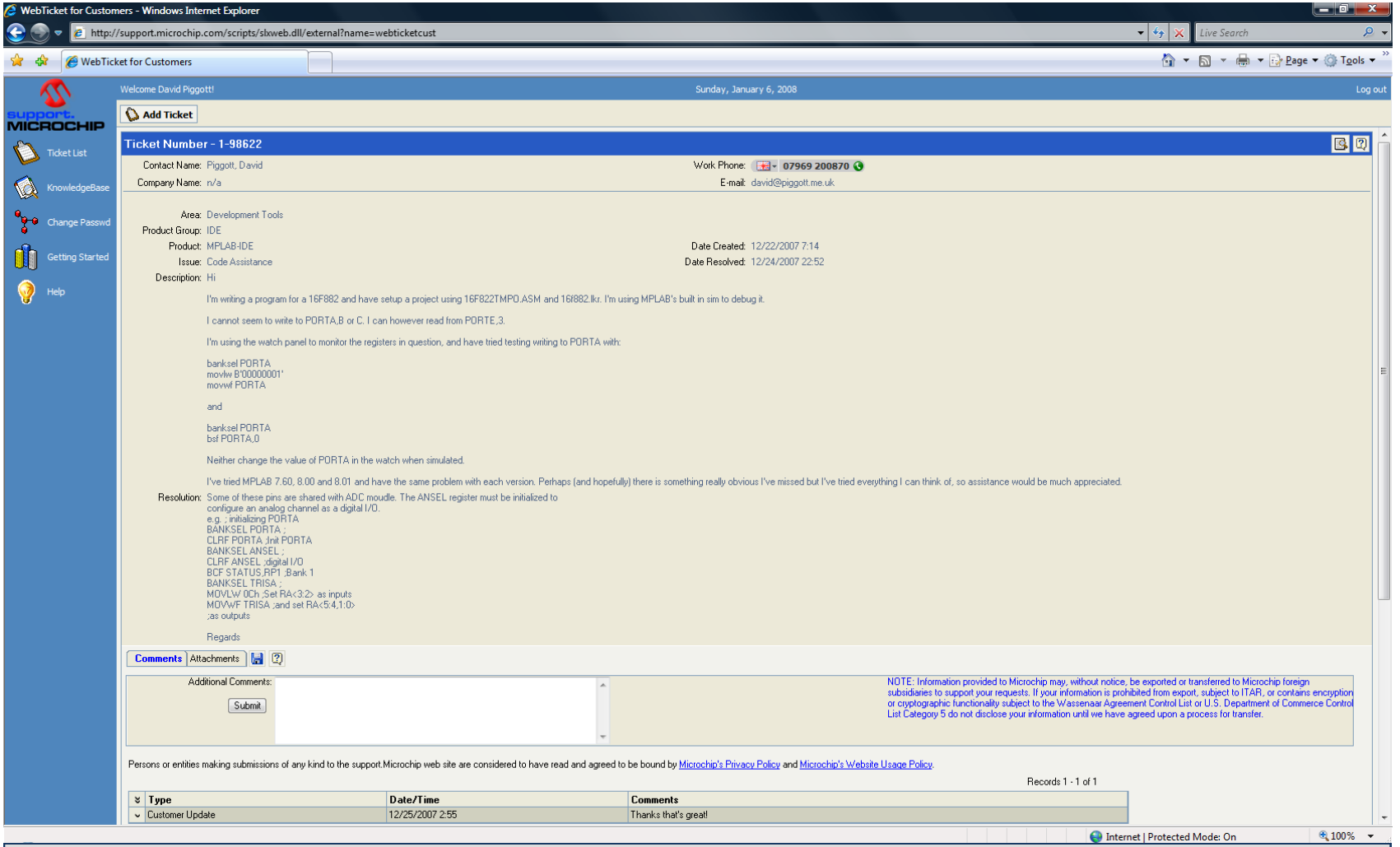


Figure 13.6 - MPLAB WebTicket System with description I gave and resolution returned.

LED Register Code Development - 0.1

This wasn't the only problem I had with outputs. When I had overcome this problem, I encountered another similar issue except that it was only effecting the six least significant bits of PORTB, which simply would not switch on. Suspecting a similar cause to that of the first problem, I navigated to the section of the datasheet on PORTB and found information shown in figure 13.7.

Therefore in order to solve the two output problems I added the code in figure 13.8 to the init(ialisation) routine:

```
banksel ANSEL
clrf    ANSEL
banksel ANSELH
clrf    ANSELH
```

Figure 13.8 - Addition to init

3.4.1 ANSELH REGISTER

The ANSELH register (Register 3-4) is used to configure the Input mode of an I/O pin to analog. Setting the appropriate ANSELH bit high will cause all digital reads on the pin to be read as '0' and allow analog functions on the pin to operate correctly.

The state of the ANSELH bits has no affect on digital output functions. A pin with TRIS clear and ANSELH set will still operate as a digital output, but the Input mode will be analog. This can cause unexpected behavior when executing read-modify-write instructions on the affected port.

Figure 13.7 - ANSELH Register information

It would be extremely counterproductive for me to spend my time documenting how the register code works instruction by instruction, so three more items remain before the documentation of 0.1 is complete:

- Necessary additions
- General flowchart
- In circuit testing

One necessary feature is missing from 0.1; support for multiple registers. To simplify the programming for this early version I omitted this. The way this works will be explained in greater detail as when I implement it.

Because I am not documenting instruction by instruction how the program works I have created a flowchart included overleaf (figure 13.9) summarising how the register code works.

LED Register Code Development - 0.1

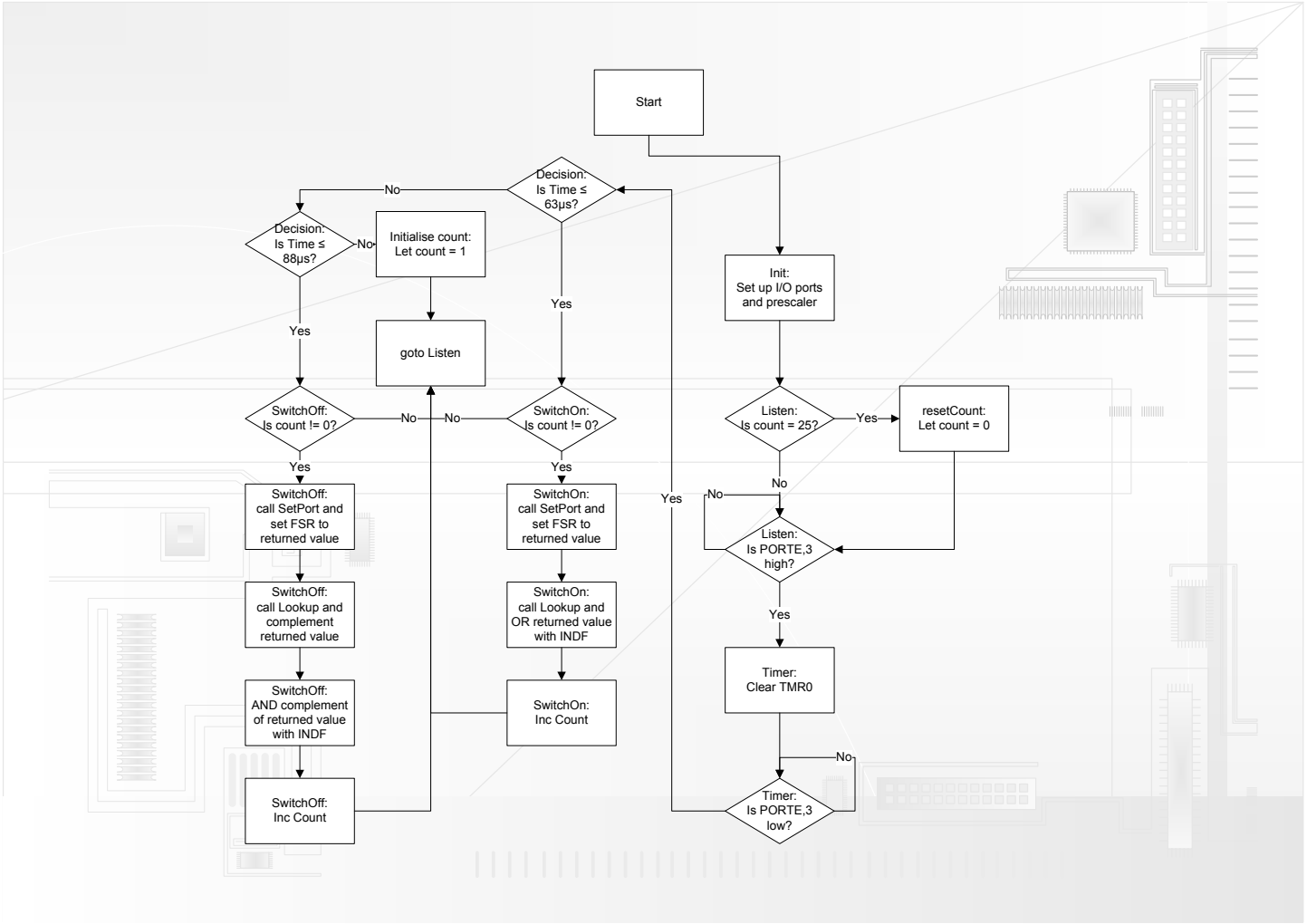


Figure 13.9 - Summary flowchart of 0.1

Problematic Programming

At this stage, I now have in my inventory the following:

- 1x PICKit 2 Programmer
- 1x PICKit 2 Demo Board with a PIC16F690
- 20x PIC16F882s (they were transported by air so arrived within a week of ordering!)
- 1x PIC18F252
- 1x Register Development PCB (produced from the respective artwork, version 0.2) (tracked for a PIC16F882)
- Some experience with assembly code, as I have looked at the example programs included with the PICKit 2

Prior to creating the register development PCB, I had successfully programmed (i.e. had the PICKit 2 software communicate successfully with the chip):

A PIC16F690 in the demo board
and
a PIC18F252 in breadboard

However, when I attempted to program a PIC16F882 in breadboard, I received an error message (see figure 14.0). After having checked (more than three times) my interpretation of the programming requirements, the schematic, the artwork, the actual PCB, and all the necessary voltages (V_{pp}, ICSPData, ICSPClock, V_{dd} and V_{ss}), I was pretty convinced that the issue was not a hardware issue.

However, based on the fact that the programmer continued to communicated successfully with the PICKit 2 Demo board, it seemed that the programmer

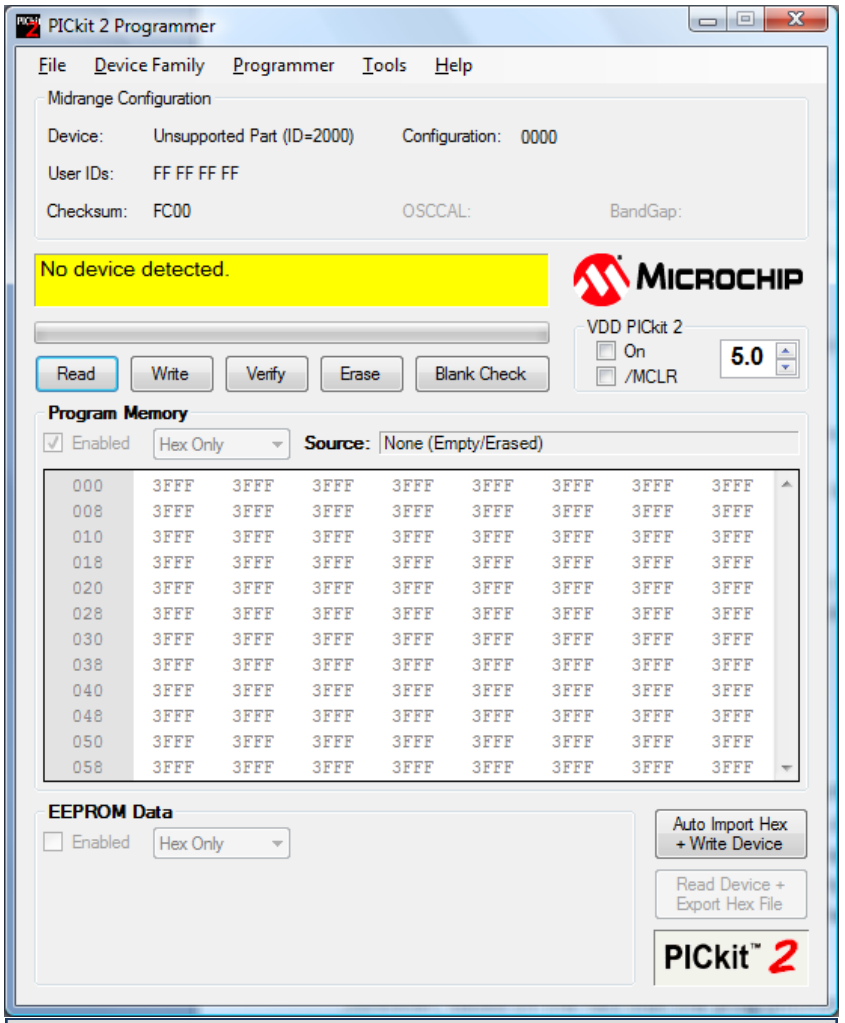


Figure 14.0 - Programming Error

Problematic Programming

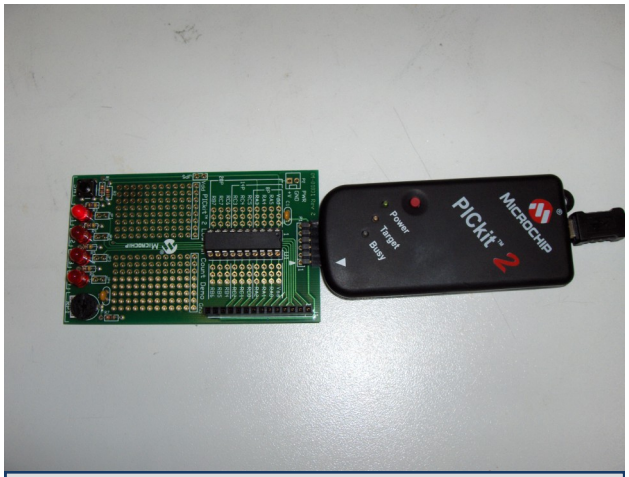


Figure 14.1 - PICKit 2 Demo Board

and software could not be the problem either. As a last check, I also used an oscilloscope to check a 30KHz signal on the ICSP clock and data lines (the PICKit 2 software has a trouble-shooting feature that allows you to output a square wave to these), and found these to be functioning correctly too.

The next step after having verified that all the hardware was correctly functioning was to check the Microchip website and knowledge base to see if this has been a common problem. In doing so I found that there is a newer version of the programmer software (2.11 was included on the CD, 2.40 was now available on the website) and firmware available for download than that included on the CD with the PICKit 2. I downloaded and installed these and found this solved the problem! To the left and below are pictures of the PICkit 2 demo board, breadboard, and register development PCB (see 'Unit Register Artwork 0.1') that I tried programming in:

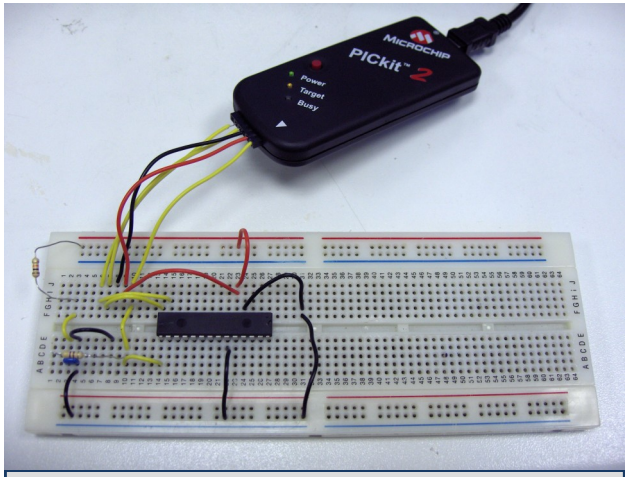


Figure 14.2 - Breadboard setup

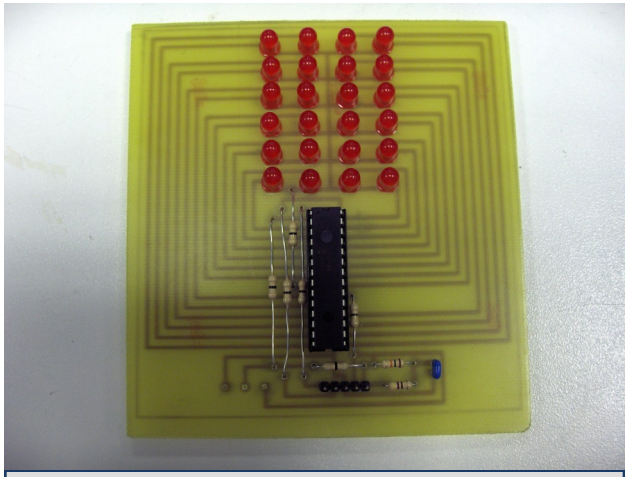


Figure 14.3 - Register development PCB

Development Dependencies

One of the challenges with this project is the issue of modelling and prototyping due to the vast quantity of LEDs it will use and PICmicro registers. Even when developing the LED register controller code, I had difficulty programming the PIC16F822 in breadboard and so resorted to creating an artwork and PCB solely for the purpose of developing the program.

Developing the main game logic code will be yet more challenging. In order to test it properly I will really need the hardware it will be running on available to me, so I can see where the seeds are (i.e. which LEDs are lit) and interact with the program by means of the pit buttons. The point I'm making is that before I can do much/any work on the master controlling code, I will need to create the schematics and artworks and produce the PCBs, so I can test it in the target application.

I have already breadboarded basic PICmicro circuits successfully aswell as created the register development board so am familiar with the operating circuits for PICmicros. This is fortunate because it will not be practical to breadboard the artworks and thus PCBs I will be producing due to their shear size and component count. For the same reasons, it is very important that I get the PCBs right first time as any errors could be very costly both in terms of time and money. I shall therefore endeavour to implement as much error checking into my work as possible when I design the schematics and artworks.

With the reasons explained, it is now that I can say I have not adhered to the development timeplan that is figure x.x. My actual use of time is explained in the section at the end titled "Time Usage".

Unit Register Schematic 0.1

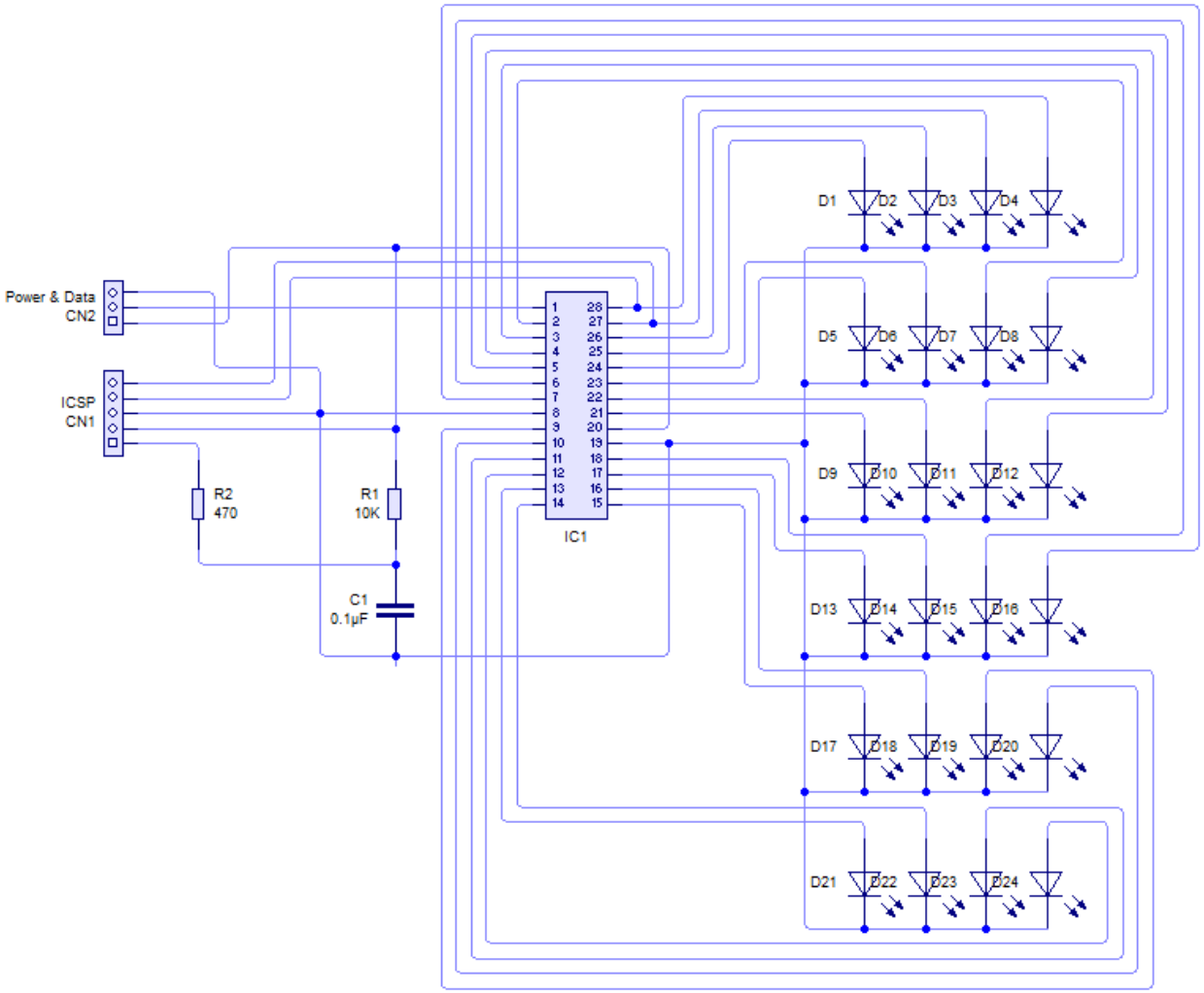


Figure 15.0 - LED Register Schematic

Figure 15.0 shows the LED register schematic. Included in this version is the ICSP (in circuit serial programming) setup, since this schematic will be the base for my program development.

Specifically which pin on the PIC16F882 each LED connects to is not of any significance, and this is likely to change when I create the art-work, to make the routing more efficient. However it will mean that I will have to alter the program.

D3 and D4 are connected to pins 27 and 28 respectively; which also function as the ICSP clock and data lines during programming.

Unit Register Artwork 0.1

Figure 16.0, Unit Register Artwork 0.1 is the artwork for a single LED register and is based on LED register schematic 0.1. It is the artwork I created the register development PCB (figure 13.3) from and that I am using for register code development. Because it was designed for development work it includes ICSP connectivity; the production register artworks will not. For the same reason there is no on board power regulation as it is powered by the PICKit 2 programmer when connected. Figure 16.1 is the coding key.

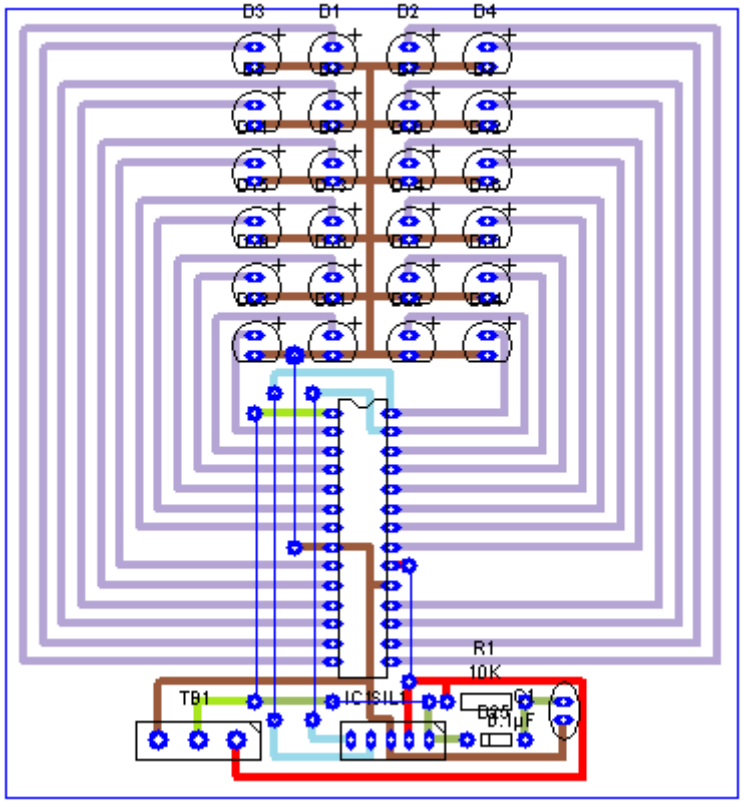


Figure 16.0 - Unit Register Artwork 0.1 - Colour coded

	Output to LED
	0v
	+5v
	Serial control line
	Programming clock and data lines
	V_{pp} Programming enable voltage

Figure 16.1 - Coding key

On completion of any artworks I create, I always colour code them. This offers several benefits:

- The link between schematic and artwork is clearer to viewers unfamiliar with the circuit (i.e. readers of this documentation).
- In going through the colour coding process, I almost always spot at least one error with the artwork and thus it is a great time saver; I would otherwise only discover these errors after having produced the PCB and found it to not work.

The colour coding process is simple:

1. I press Ctrl-A and then Ctrl-C in PCB Wizard to select the entire work and copy to the clipboard.
2. I open the Microsoft Paint program and paste the artwork into it.
3. Using the fill tool, I change the colour of every track on the board based on it's function.

While fine as a development board, I did not spend any time on space optimisation with register artwork 0.1 and so it is not suitable as a production board because it is simply too large for me to be able to meet the specification points on case size. Therefore it was necessary for me to create smaller register artworks.

Unit Register Artwork 0.2

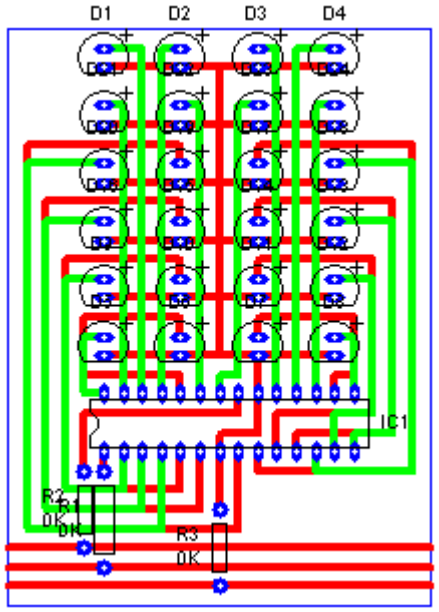


Figure 17.0 - Unit Register Artwork 0.2

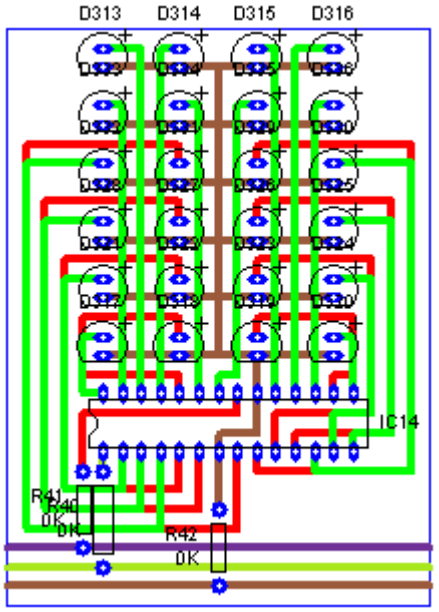


Figure 17.1 - Unit Register Artwork 0.2 - Colour coded

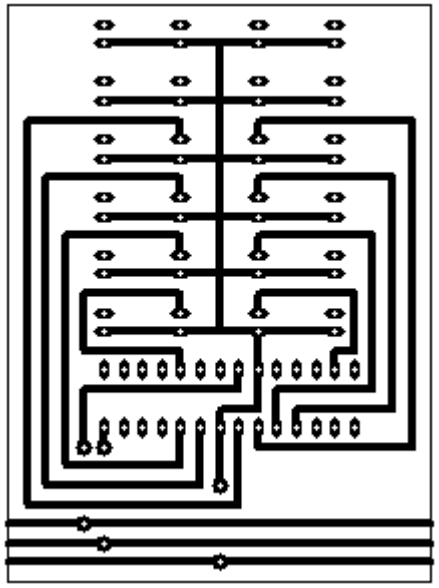


Figure 17.2 - Unit Register Artwork 0.2 - Solder side

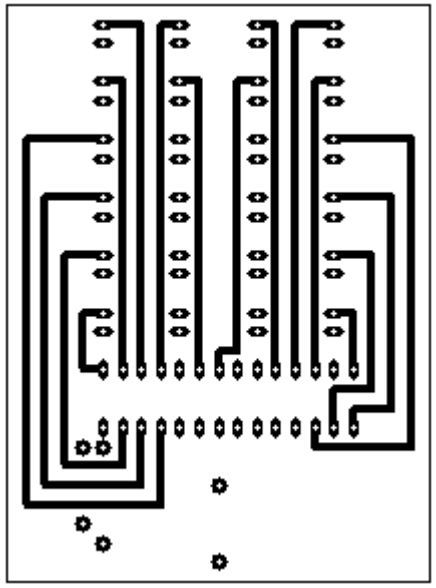


Figure 17.3 - Unit Register Artwork 0.2 - Component side

In order to make the register artworks as small as possible, I created register artwork 0.2 as a dual layer artwork, as seen above, so I could get an idea of how much of an advantage making dual layer boards would be.

	+5v		Solder side output track to LED
	0v		Component side output track to LED
	Serial control line		

Figure 17.4 - Coding key

Unit Register Artwork 0.3

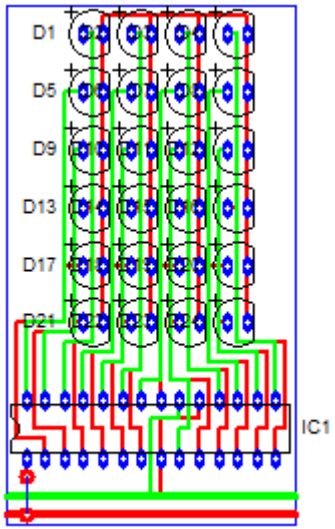


Figure 18.0 - Unit Register Artwork 0.2

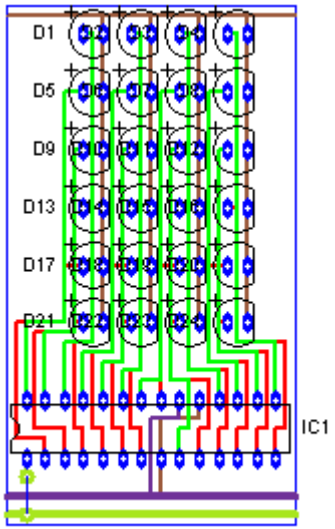


Figure 18.1 - Unit Register Artwork 0.2 - Colour coded

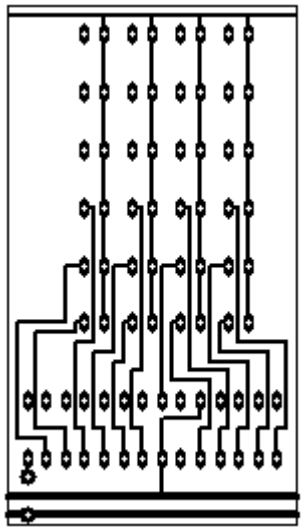


Figure 18.2 - Unit Register Artwork 0.2 - Solder side

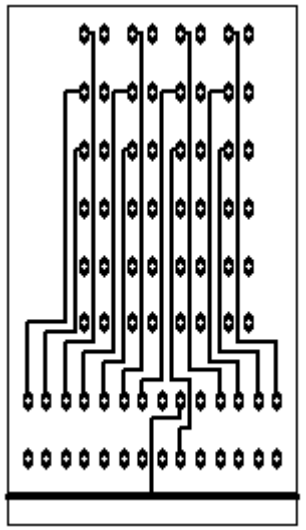


Figure 18.3 - Unit Register Artwork 0.2 - Component side

Artwork 0.3 is even more ambitious than artwork 0.2; not only is it dual layer, but I reduced the track size to 0.02” from 0.04” (default) and the track spacing to 0.05” from 0.1” (default). This enables more tracks to run parallel to each other in a given space, and for tracks to run between the pads of LEDs!

	+5v		Solder side output track to LED
	0v		Component side output track to LED
	Serial control line		

Figure 18.4 - Coding key

When I created artwork 0.2 and 0.3, I knew creating dual layer PCBs with the school equipment would be a challenge. I continued with the artwork creation anyway so I could see exactly how much smaller I could make the artworks. I will not however be producing dual layer PCBs on only one board.

Schematic & Artwork Set 0.1 - Game Logic

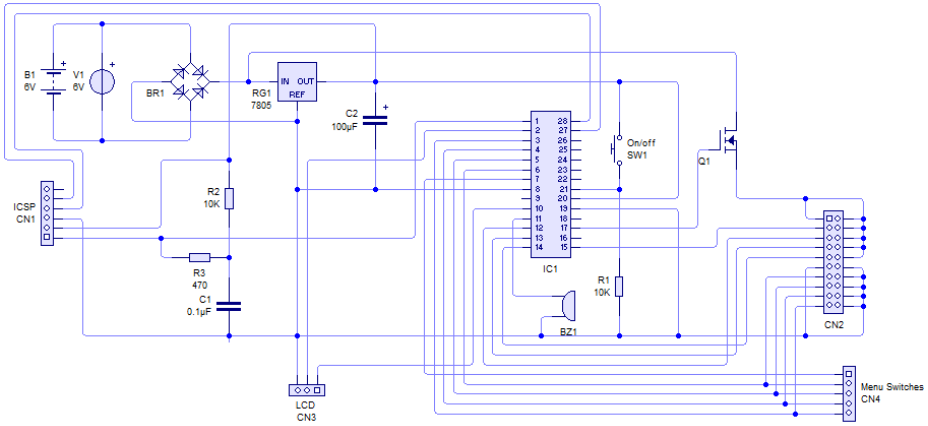


Figure 19.0 - Game Logic Schematic 0.1

As discussed in the task analysis, the board features dual power supplies; battery and low voltage DC from the mains. I have included a bridge rectifier because for their low cost (compared overall project cost) they offer protection to the other components against incorrect battery/power supply polarity. The voltage regulator on this schematic/board powers only the game logic components and peripherals such as the buzzer and LCD. The on/off switch is connected to the PICmicro and toggles the software status; on power down the game state is saved and the PICmicro enters low power sleep mode. An interrupt is set on the on/off switch port (it is connected to this specific pin for a reason as on the PIC18F252 this pin supports interrupts).
A MOSFET controls power to the pit board registers allowing them to be completely switched off when the board is off.

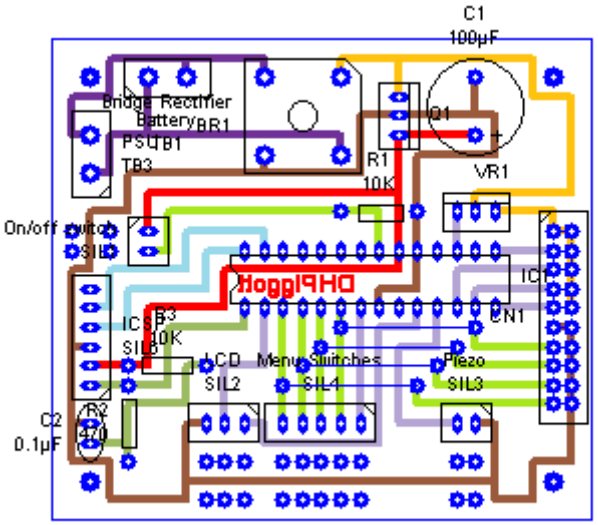


Figure 19.1 - Game Logic Artwork 0.1 - Colour Coded

	Power supply
	0v
	Unregulated +V
	+5v
	Input
	Output
	Programming clock and data lines
	V_{pp} Programming enable voltage

Figure 19.2 - Coding key

Schematic & Artwork Set 0.1 - Menu Buttons & Routing Board

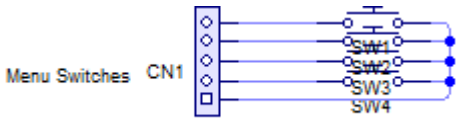


Figure 20.0 - Menu Buttons Schematic 0.1

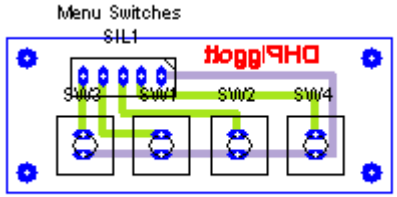


Figure 20.1 - Menu Buttons Artwork 0.1 - Colour Coded

	0v		Input
	Unregulated +V		Output
	+5v	Figure 20.2 - Coding key	

Power provided by the game logic board to the pit board registers is unregulated, so a regulator is included on this 'routing board'.

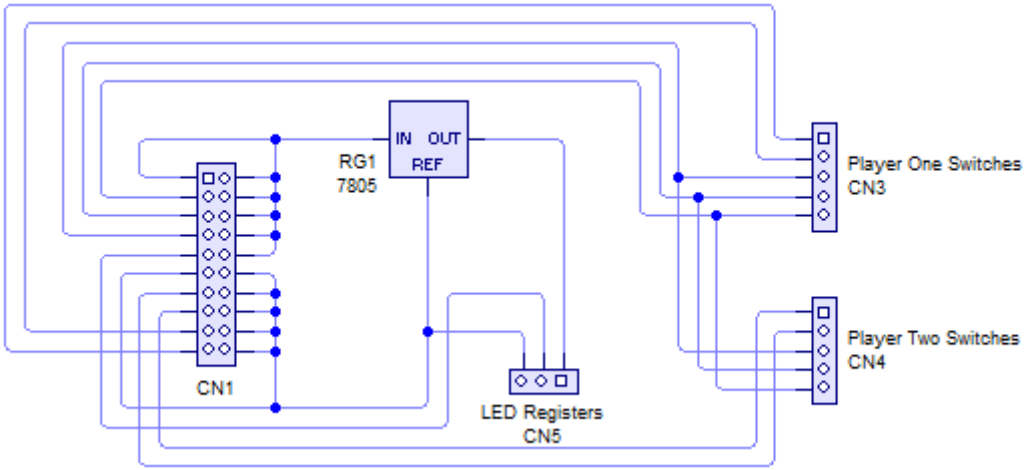


Figure 20.3 - Routing Board Schematic 0.1

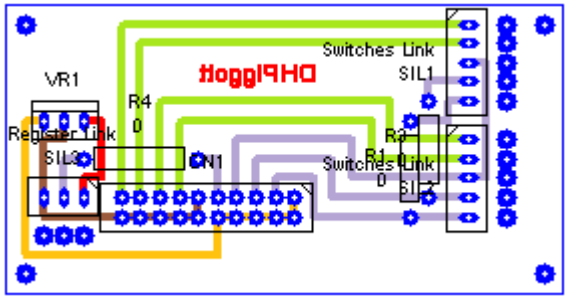


Figure 20.4 - Routing Board Artwork 0.1 - Colour Coded

Schematic & Artwork Set 0.1 - Pit Buttons & Registers

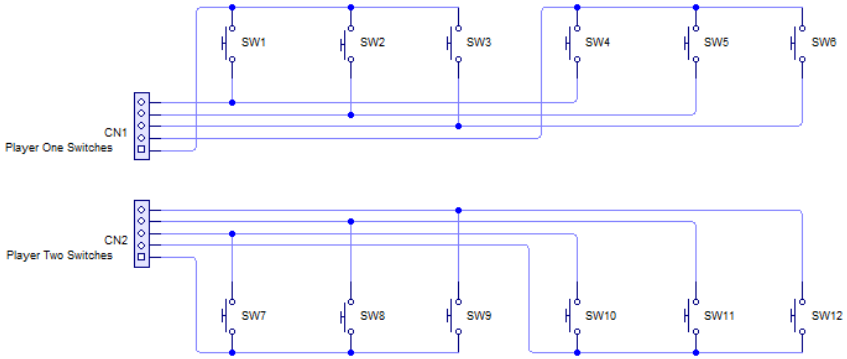


Figure 21.0 - Pit Buttons Schematic 0.1

The pit buttons (12 in number) and menu buttons (4 in number) are connected in an array setup to the game logic PICmicro, and since there are 16 in total this can be catered for with a 4 x 4 array. This is divided into two sub arrays; a 3 x 4 for the pit buttons, and a 1 x 4 for the menu buttons.

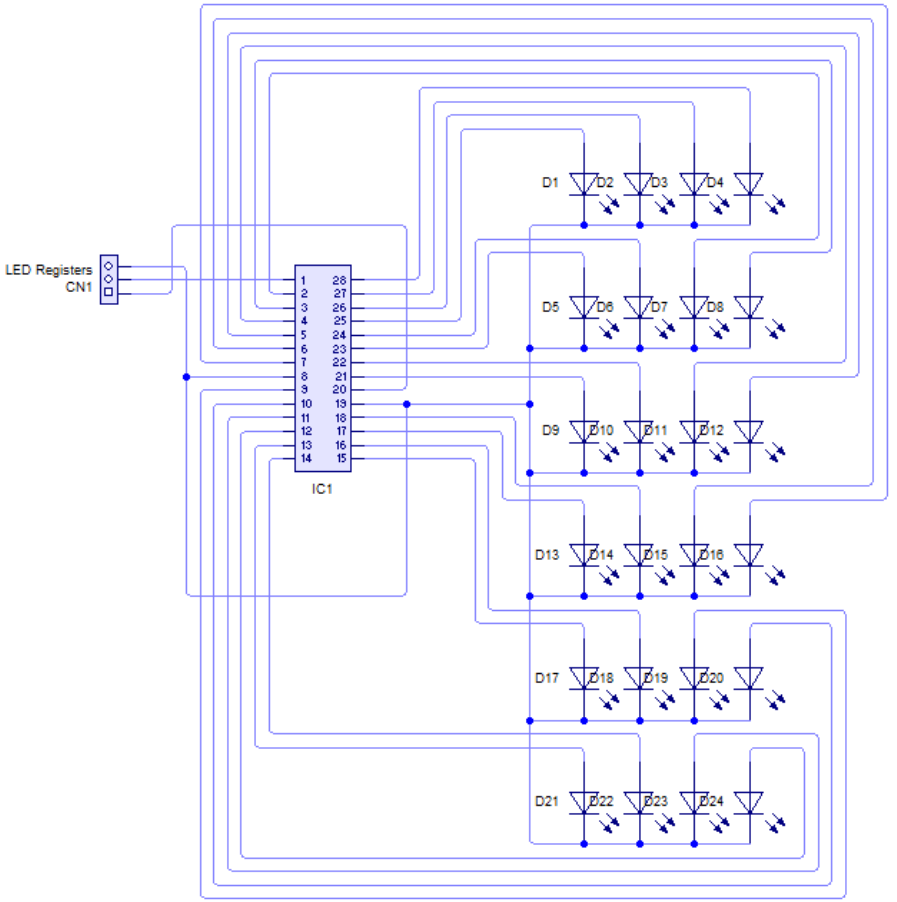


Figure 21.1 - Register Schematic 0.1

Schematic & Artwork Set 0.1 - Board Artwork Upper Layer

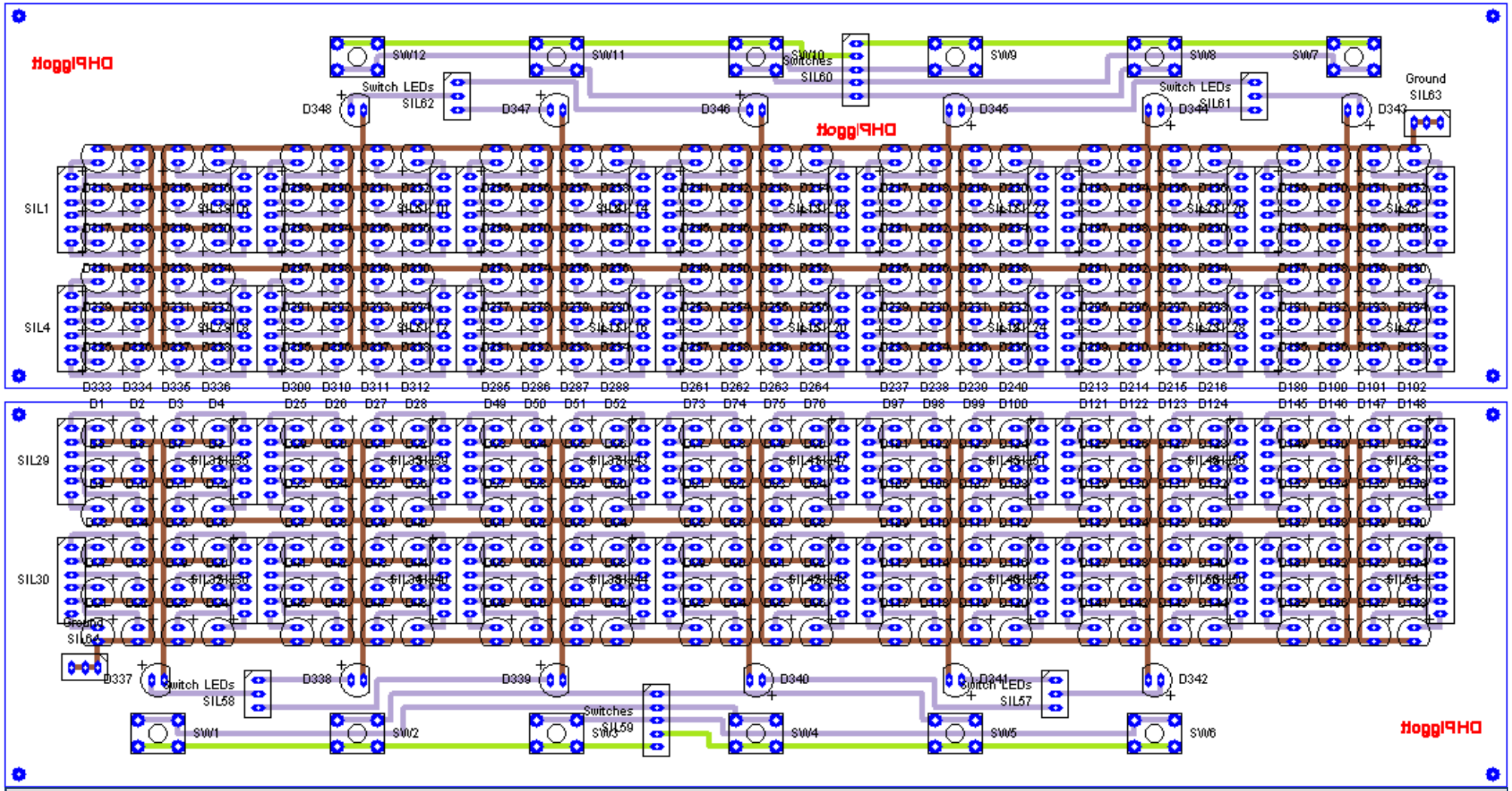


Figure 22.0 - Board Artwork Upper Layer 0.1 - Colour Coded

Schematic & Artwork Set 0.1 - Board Artwork Lower Layer

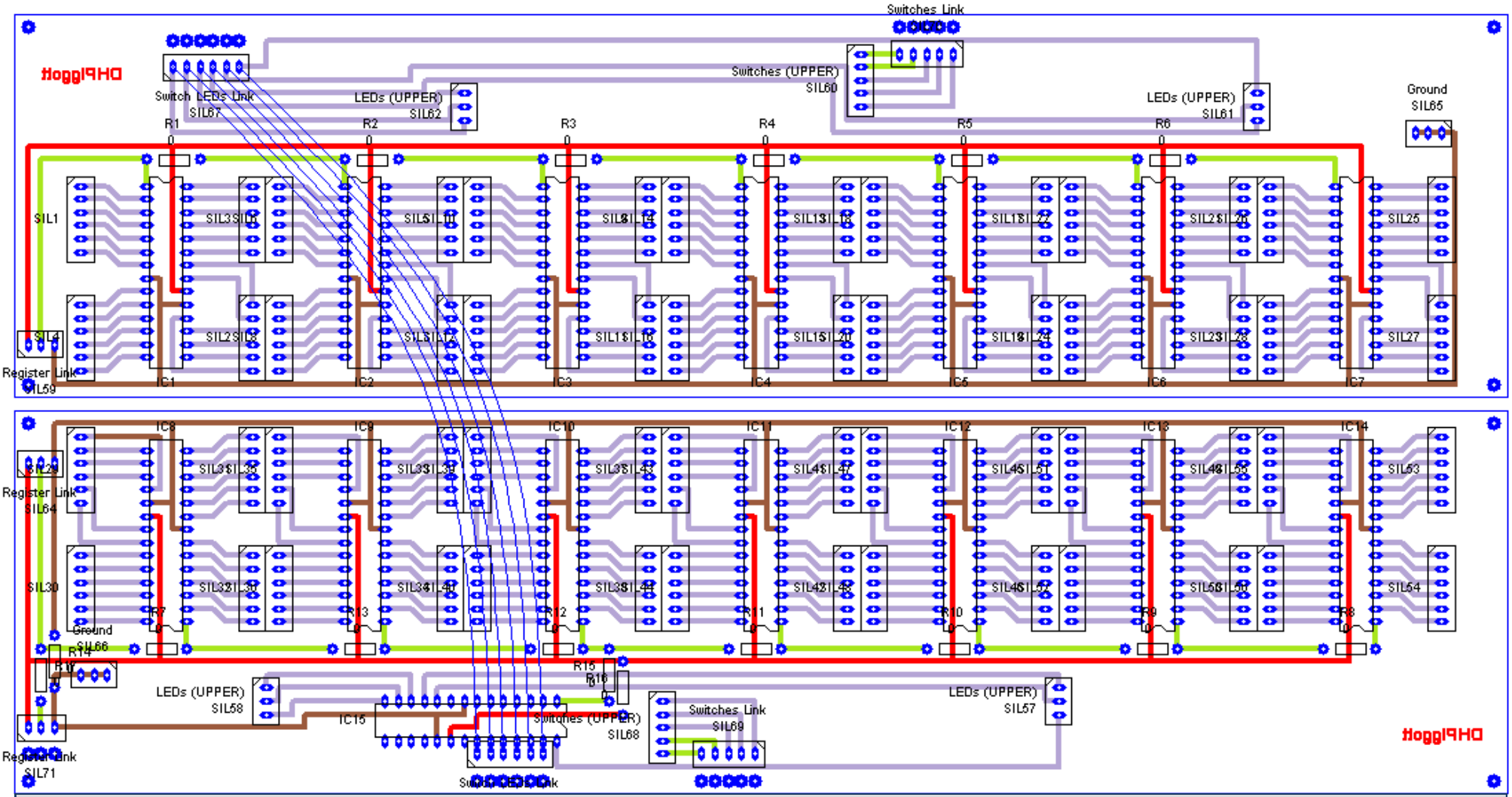


Figure 23.0 - Board Artwork Lower Layer 0.1 - Colour Coded

Schematic & Artwork Set 0.1 - Evaluation

	0v
	+5v
	Input
	Output

Figure 24.0 - Coding key

Figure 24.0 is the coding key to be used with the board artwork upper and lower layers.

As with unit register artworks 0.2 and 0.3, I will not be producing PCBs from schematic and artwork set 0.1. However for the sake of documentation completeness they are included here. The reason for not producing PCBs from them is that the development of the schematics and artworks was a parallel process; that is to say, the actual circuit changed while I was routing the artworks and so the layouts are not optimal, so much so that I had to create a 'routing board' to facilitate interconnections between the pit boards and the game logic board. The imperfections can be summarised as follows:

- Insufficient thought given to the power supply and power management
- Non-optimal use of space
- An unnecessary 'routing board' was required because I designed the pit boards after the game logic board
- Difficulty could result when trying to connect the upper and lower pit boards due to insufficient planning of interconnection (lots and lots and lots of SIL connections!)
- Insufficient thought given to the positioning of the PCBs within the case

I intend to fix all the imperfections in set 0.1 when I create set 0.2 as I hope to create the PCB set from this. However one significant development with artwork set 0.1 is that the board artworks are suitably compact without having resorted to conventional dual layer boards, or impractical track widths and spacing.

Also worth noting at this point is that all artworks are included in this documentation at the same scale as they would be on PCB.

Case Plan - Component Locations Draft

Plan View

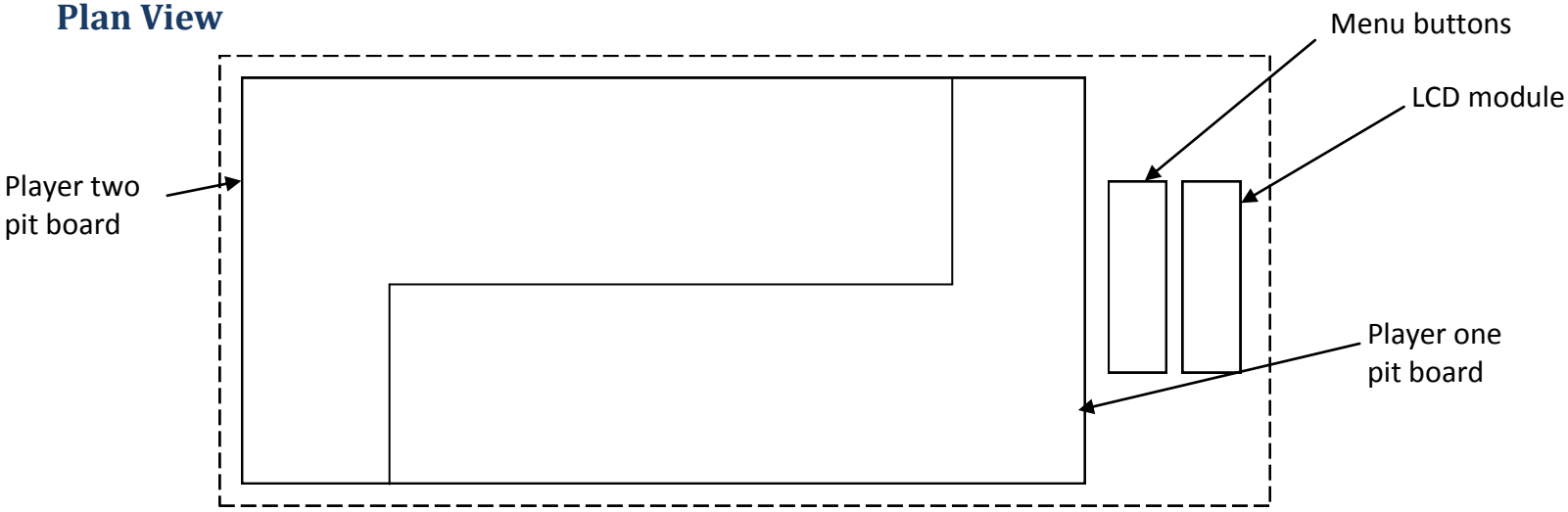


Figure 25.0 - Component Locations Draft - Plan View

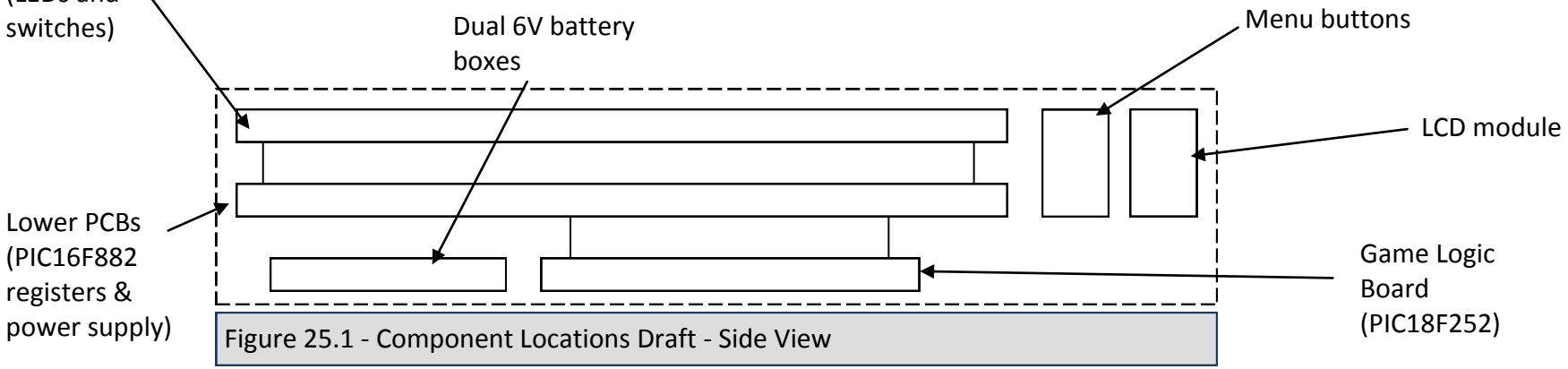


Figure 25.1 - Component Locations Draft - Side View

Schematic & Artwork Set 0.2 - Game Logic & Button Schematics

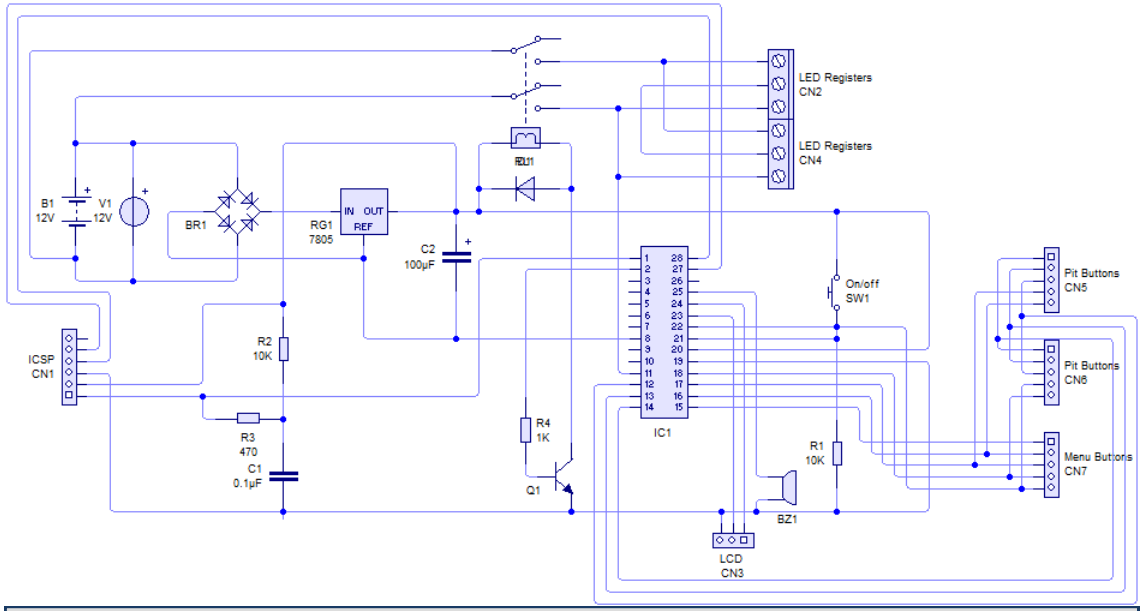


Figure 26.1 - Game Logic Schematic 0.2

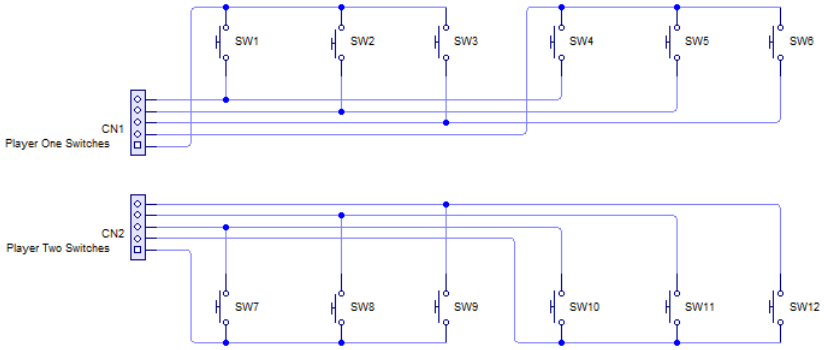


Figure 26.3 - Pit Buttons Schematic 0.2

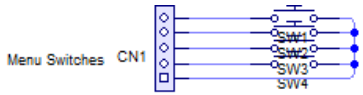


Figure 26.0 - Menu Buttons Schematic 0.2

As a system wide change, the power supply is now split up into parts for each board; the game logic board has its own bridge rectifier and voltage regulator, designed to provide up to 1A. As with game logic schematic 0.1, the LCD power is controlled directly by the PIC with no intermediate transistor, based on the fact the current requirement for the LCD module is specified as 5mA.

More of the menu/pit button routing is done on the game logic board, negating the need for a separate routing board.

As part of the power supply redesign, the LED registers have their own onboard power systems designed to accept either polarity, so in order to avoid having large bridge rectifiers on the game logic board, power to the register boards is controlled by a DPDT relay (because in this setup it is non polar, whereas a MOSFET as used before would be, unless I had used two in inverse parallel, but having not done this before I didn't want the risk of it not working).

Because of the voltage drops across bridge rectifiers and voltage regulators, and because of the high estimated current consumption of the finished product, I have upgraded the power supply to 12V (transformed mains or batteries).

Schematic & Artwork Set 0.2 - LED Register Schematic

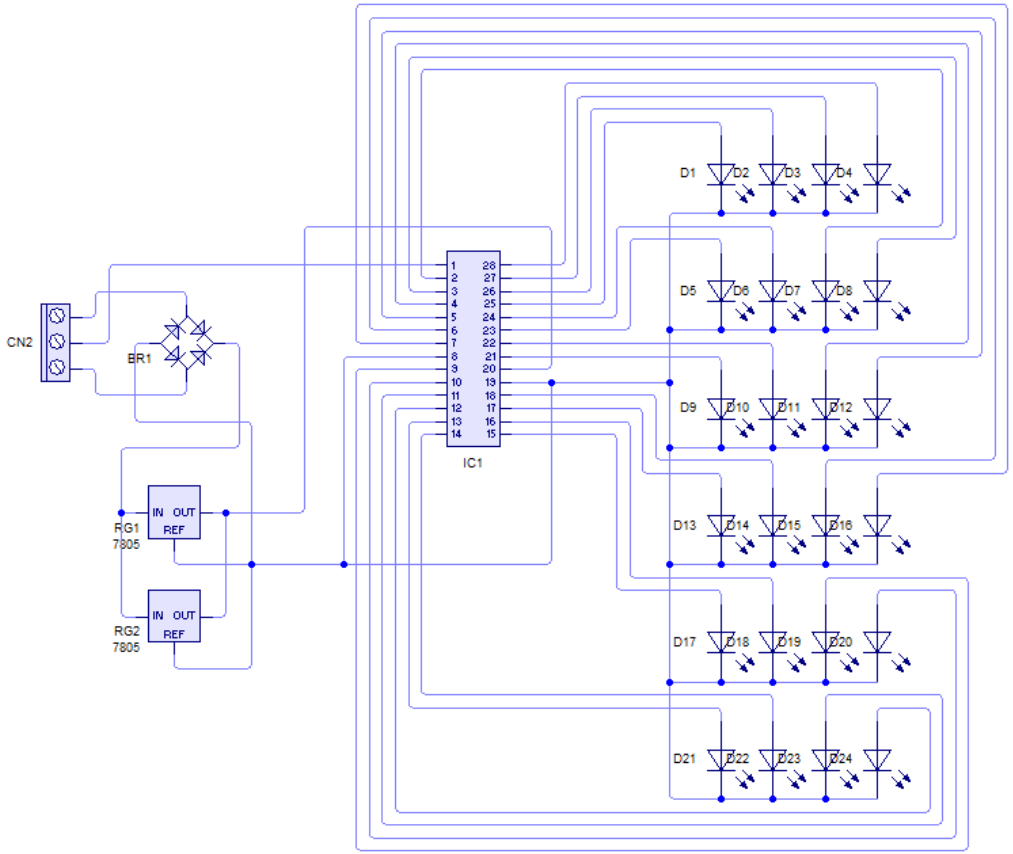


Figure 27.0 - LED Register Schematic 0.2

As with schematic set 0.1, the pit boards will be split by player number; one for player one, one for player two. Based on my estimate of 4.2A (168 x 25mA = 4.2A), I have provisioned for 4A per player; although only 24 LEDs should only ever be on in gameplay, it makes sense to provision for more should I have spare time/program memory one the core game logic is programmed to add some visual effects for when a player wins etc.

This is provided by means of a 8A bridge rectifier on each board, and a pair of 2A 7805 5V regulators in parallel. On each board there are then seven pits powered from this; six standard (red or blue depending on player) and one mancala (white).

The schematic in figure 27.0 shows only one register, and the mapping between LED and PICmicro pins are arbitrary (that is to say, they differ on the artworks, and I chose them based on most efficient routing).

Schematic & Artwork Set 0.2 - Game Logic & Menu Buttons

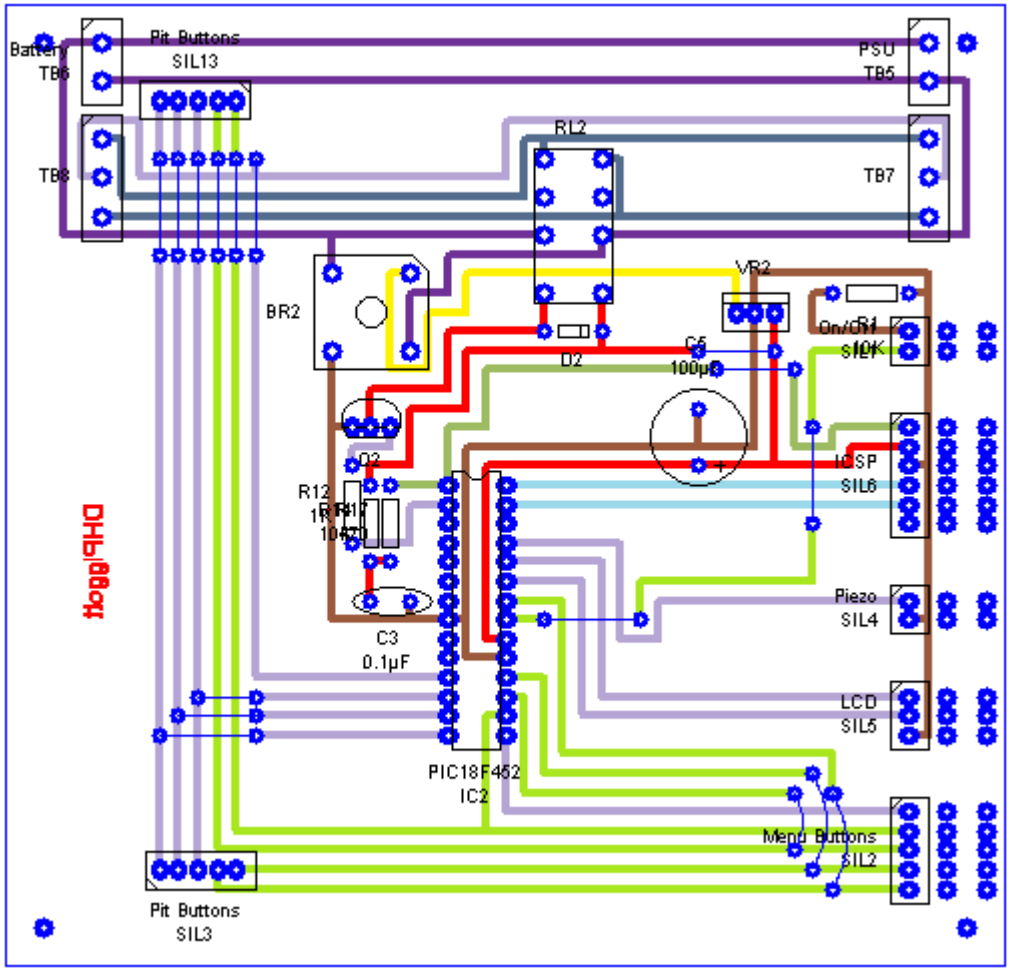
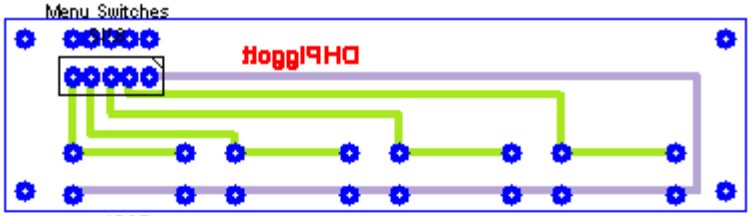


Figure 28.2 - Game Logic Artwork 0.2



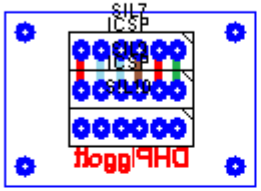


Figure 28.1 - ICSP Case mounting socket

Figure 28.0 - Menu Buttons Artwork 0.2

	Power Supply
	Relay Controlled Register Power Supply
	0v
	Unregulated +V
	+5v
	Input
	Output
	Programming clock and data lines
	V_{pp} Programming enable voltage

Figure 28.3 - Coding key

Schematic & Artwork Set 0.2 - Player One Lower Artwork

	Relay Controlled Register Power Supply
	0v
	Unregulated +V
	+5v
	Input
	Output

Figure 29.0 - Coding key

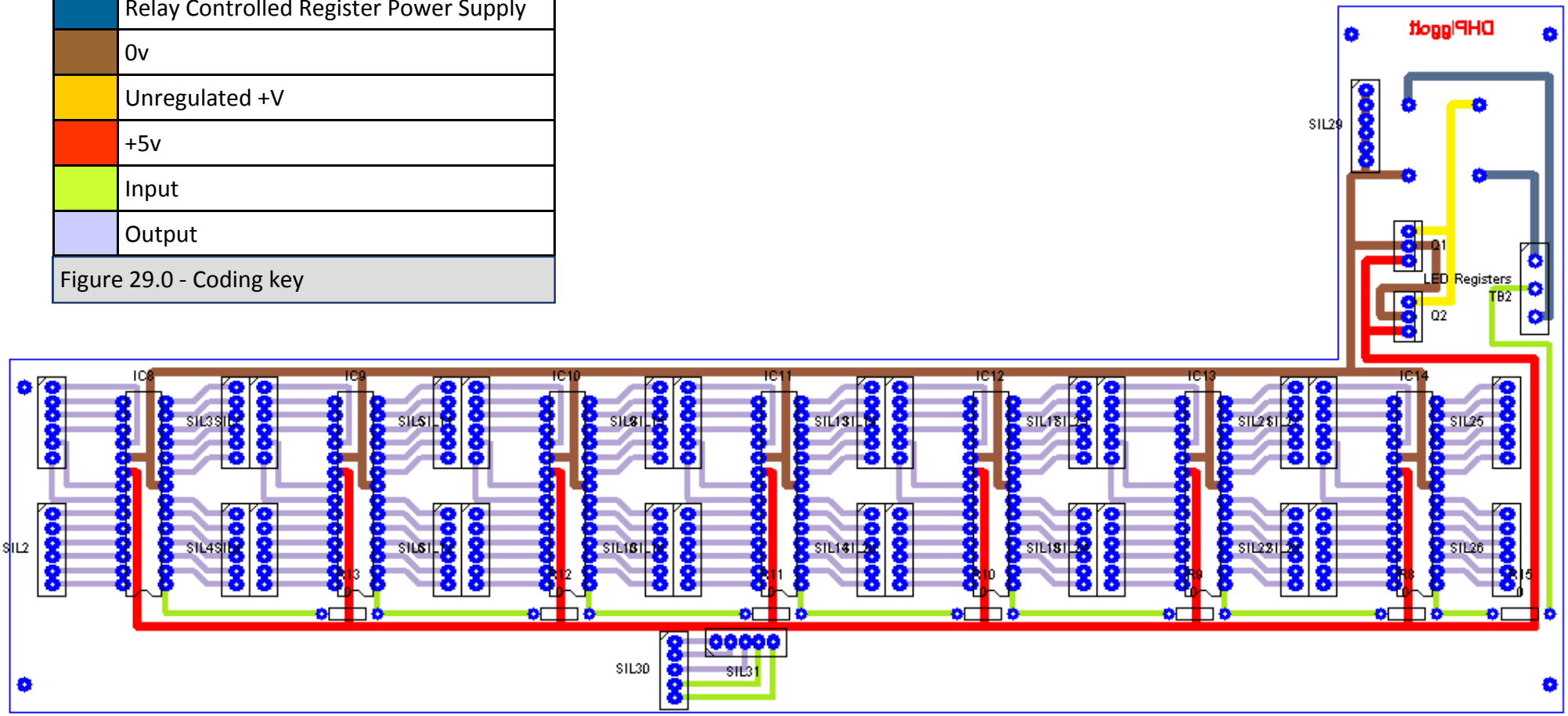


Figure 29.1 - Player One Lower Artwork 0.2

Schematic & Artwork Set 0.2 - Player One Upper Artwork

	0v
	Input
	Output

Figure 30.0 - Coding key

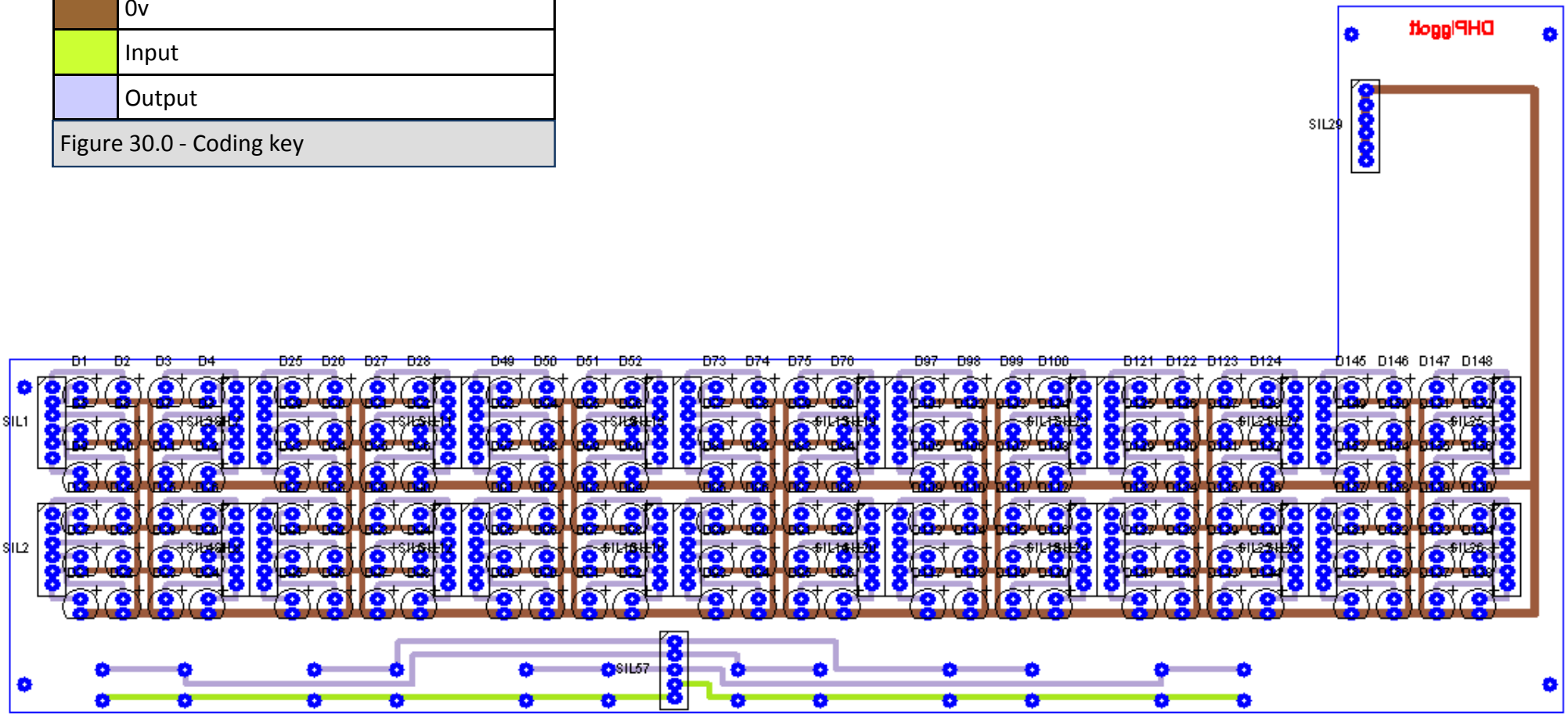
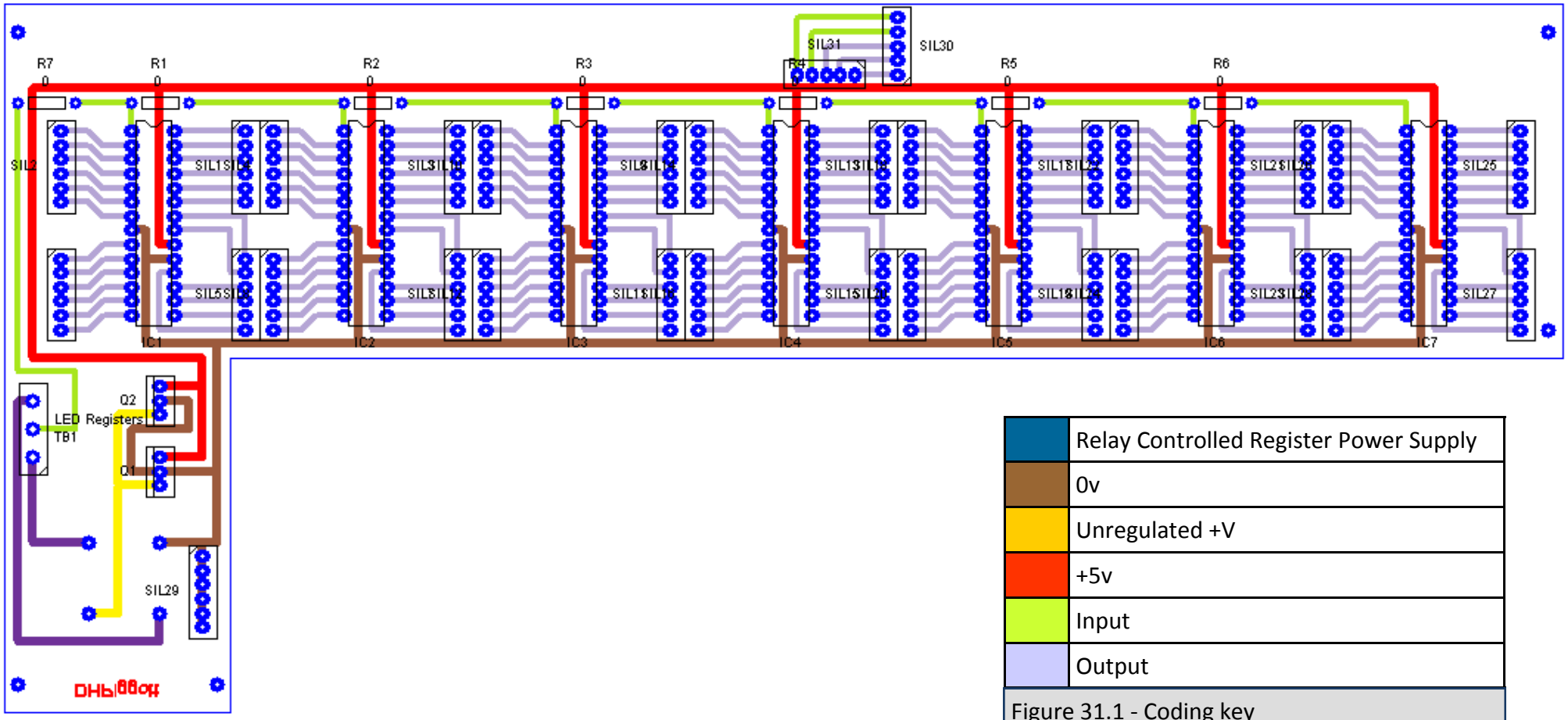


Figure 30.1 - Player One Upper Artwork 0.2

Schematic & Artwork Set 0.2 - Player Two Lower Artwork

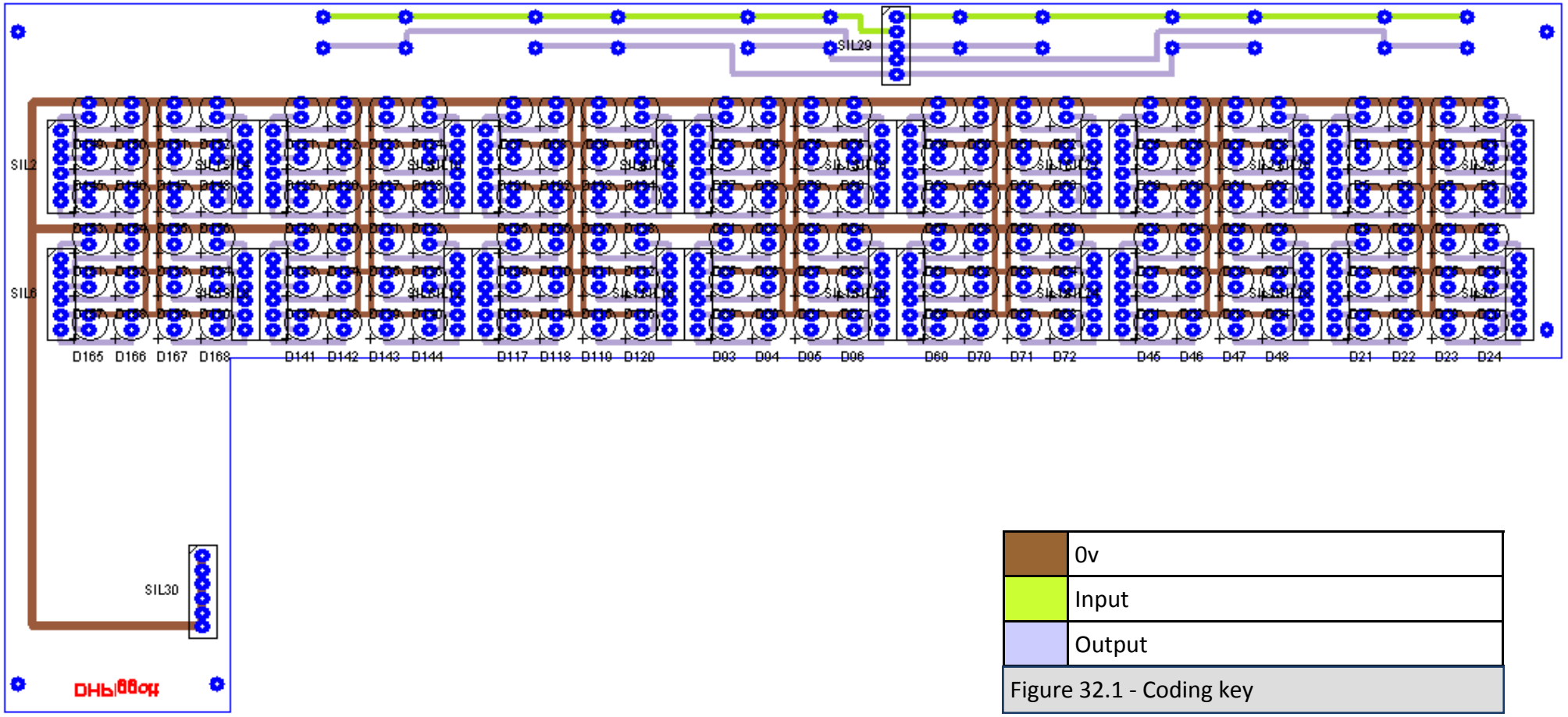


	Relay Controlled Register Power Supply
	0v
	Unregulated +V
	+5v
	Input
	Output

Figure 31.1 - Coding key

Figure 31.0 - Player Two Lower Artwork 0.2

Schematic & Artwork Set 0.2 - Player Two Upper Artwork



	0v
	Input
	Output

Figure 32.1 - Coding key

Figure 32.0 - Player Two Upper Artwork 0.2

Schematic & Artwork Set 0.2 - Evaluation

I noted the flaws with schematic & artwork set 0.1 as being the following:

- Insufficient thought given to the power supply and power management
- Non-optimal use of space
- An unnecessary 'routing board' was required because I designed the pit boards after the game logic board
- Difficulty could result when trying to connect the upper and lower pit boards due to insufficient planning of interconnection (lots and lots and lots of SIL connections!)
- Insufficient thought given to the positioning of the PCBs within the case

Resolutions:

- I have now fixed the majority of these. Power to the register boards is controlled by a relay, which is driven by a transistor from the PIC. Once at the register boards it passes through a bridge rectifier (to protect against incorrect polarity) and then a pair of 2A rated 7805 5V regulators in parallel, allowing for current of up to 4A; there are 168 LEDs on each board, and assuming current flow of 25mA per LED this is 4200mA or 4.2A. Although in gameplay only 24 LEDs will ever be lit at once, it is good practise to provide greater capacity. Because of the voltage drops across the bridge rectifiers and voltage regulators, 2x 4x AA battery boxes will be required (12V)
- I have endeavoured to make the register boards as small as is practical, and this has proved sufficient as it will allow the case to be sub laptop-size. I did not invest a great deal of time in making the game logic board as small as possible because I would still be constrained by the size of the register boards, so it would be a wasted effort.
- No 'routing board' is required with this set, so that issue has been fixed.
- I haven't been able to reduce the number of PCB interconnects significantly because they are required in this dual layer setup, and any other design would be orders of magnitude larger.
- I have planned out the location of all boards and components in the case, and designed the PCBs to work with this design (documented prior to this schematic and artwork set 0.2 section).

After having exposed, developed and etched the game logic board, the two lower boards, and having completed drilling the game logic and menu button boards, I realised that the pads I had used in the artworks were too small, because as the drill bit wore out, the pads were getting more and more destroyed each time. I remedied this by increasing the pad sizes and starting again. An unfortunate waste of time, resources and money, but I believe it makes more sense to stop and start again here than later down the line after having spent more time on it. Rather than calling this one version 0.3, I have updated 0.2 because I consider this a minor fix, and it saves an extra unnecessary ten pages.

Pit numbers, LED Mappings & Switch Array Addresses

14	13	12	11	10	9	8	
	1	2	3	4	5	6	7

Figure 33.0 - Pit Numbers table

1 - RC4	2 - RC7	3 - RC3	4 - RC2
5 - RC6	6 - RC5	7 - RC1	8 - RC0
9 - RB1	10 - RB0	11 - RA6	12 - RA7
13 - RB3	14 - RB2	15 - RA5	16 - RA4
17 - RB5	18 - RB4	19 - RA3	20 - RA2
21 - RB7	22 - RB6	23 - RA1	24 - RA0

Figure 33.1 - LED Mappings table

	RC1	RC2	RC3	RC4
RC5	Pit 10	Pit 11	Pit 9	Menu 4
RC6	Pit 14	Pit 12	Pit 13	Menu 3
RC7	Pit 3	Pit 1	Pit 2	Menu 2
RB1	Pit 6	Pit 4	Pit 5	Menu 1

Figure 33.2 - Switch array addresses table

In preparation for the game logic and register programming, I have allocated numbers to the pits; player one's are 1-7, with seven being the mancala, and player two's are 8-14, with fourteen being the mancala. This is shown in figure 33.0.

This was also necessary for the LEDs on the pit boards, so that register code 0.2 can be written such that the first bit turns on the first LED, and the second bit turns on the second LED etc. (otherwise the ordering would appear random due to the tracking and arrangement of IO on the 16F882).

The pit board buttons and menu buttons are arranged in a four by four array. A three by four segment of this is for the pit buttons (twelve in total) and one by four segment for the menu buttons (four in total). Figure 33.2 shows the address of each button.

LED Register Code Development - 0.2

Figure 34.0 - Register Code 0.2

```
;*****************************************************************************
;                                                                            *
; Filename:         16F882 Register Code                                     *
; File Version:     0.2                                                      *
;                                                                            *
; Author:           David Piggott                                            *
; Company:          piggott.me.uk                                            *
;                                                                            *
;*****************************************************************************

  LIST   p=16F882       ; list directive to define processor
  #INCLUDE <p16F882.inc>      ; processor specific variable definitions

  __CONFIG _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF & _BOR_OFF &
_CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON & _WDT_OFF &
_INTRC_OSC_NOCLKOUT

  __CONFIG _CONFIG2, _WRT_OFF & _BOR40V

;' __CONFIG' directive is used to embed configuration word within .asm file.
; The lables following the directive are located in the respective .inc file.
; See data sheet for additional information on configuration word settings.
; Note that the _DEBUG_ON is changed by selecting a debugger or programmer.
; There isn't much advantage to setting it in the code.

;-----------------------------------------------------------------------------
;
; Variable Definitions
;
; Available Data Memory (also RAM) is divided into 4 Banks.  Special Function
; Registers, General Purpose Registers, and Access Registers are shown below:
;
;       Bank 0     Bank 1     Bank 2     Bank 3
; SFR   0x00-0x1F  0x80-0x9F  0x100-0x10F  0x180-0x18D
; GPR   0x20-0x7F  0xA0-0xBF  -          -
; ACCESS  0x70-0x7F  0xF0-0xFF  0x170-0x17F  0x1F0-0x1FF
;
; The Access Bank Registers are mutual across each address space.
;
;-----------------------------------------------------------------------------

; UDATA declares a section of uninitialized data
VARIABLES  UDATA      ; VARIABLES is the name of the section of memory
pulseLength RES  1    ; uninitialized data, placed by linker in GPR's.
pulseCount  RES  1
pitID       RES  1
pitCount    RES  1
maskComp    RES  1
```

LED Register Code Development - 0.2

```
; UDATA_SHR declares a section of shared (across all banks) uninitialized data
INT_VAR   UDATA_SHR     ; INT_VAR is the section name in Access RAM
w_temp    RES   1       ; variable used for context saving
status_temp RES   1     ; variable used for context saving

;-----------------------------------------------------------------------
; EEPROM INITIALIZATION
;
; The 16F882 has 128 bytes of non-volatile EEPROM, starting at address 0x2100
;
;-----------------------------------------------------------------------

        ORG  0x2100
        DE   0x00, 0x01, 0x02, 0x03

;-----------------------------------------------------------------------
; RESET VECTOR
;-----------------------------------------------------------------------

RESET_VECTOR   CODE   0x0000
        goto   START        ; When using debug header, ICD2 may not stop
                            ; on instruction 0 during reset.

;-----------------------------------------------------------------------
; INTERRUPT SERVICE ROUTINE
;-----------------------------------------------------------------------

INT_VECTOR     CODE  0x0004 ; Interrupt Vector Location
        goto   INTERRUPT

PROGRAM    CODE

INTERRUPT               ; Relocatable Interrupt Service Routine

;       Context saving for ISR
        MOVWF  w_temp       ; save off current W register contents
        MOVF   STATUS,w     ; move status register into W register
        MOVWF  status_temp  ; save off contents of STATUS register

;-----------------------------------------------------------------------
; SAMPLE INTERRUPT SERVICE ROUTINE
;-----------------------------------------------------------------------

;       If the interrupt came from the timer, execute the TMR0 interrupt
;       service routine. This may be removed in addition to the sample
;       program below.
;       BTFSC   INTCON, T0IF      ; Uncomment this line to test sample code
;       CALL    TMR0_ISR          ; Uncomment this line to test sample code

;-----------------------------------------------------------------------
; END OF SAMPLE INTERRUPT SERVICE ROUTINE
```

LED Register Code Development - 0.2

```
;-----------------------------------------------------
;
;       Restore context before returning from interrupt
;       MOVF   status_temp,w    ; retrieve copy of STATUS register
        MOVWF  STATUS           ; restore pre-isr STATUS register contents
        SWAPF  w_temp,f
        SWAPF  w_temp,w         ; restore pre-isr W register contents
        RETFIE                  ; return from interrupt

;-----------------------------------------------------
; MAIN PROGRAM
;-----------------------------------------------------

START

;-----------------------------------------------------
; SET OSCILLATOR TO FACTORY FREQUENCY AND CLEAR GPR's
;-----------------------------------------------------

        ERRORLEVEL -302   ; disable warning accessing register not in bank 0
        BANKSEL OSCTUNE   ; select bank 1 using mpasm macro
        MOVLW  0x00       ; set oscillator to factory calibrated frequency
        MOVWF  OSCTUNE
        BANKSEL STATUS    ; select bank 0 using mpasm macro... STATUS is arbitrary
        ERRORLEVEL +302   ; re-enable warning accessing register not in bank 0
CLEAR_RAM               ; code sequence initializes all GPR's to 0x00
                ; unnecessary if initalizing all variables before use
        MOVLW  0x20       ; initialize pointer
        MOVWF  FSR        ; to RAM
NEXT
        CLRF   INDF       ; clear INDF register
        INCF   FSR, F     ; inc pointer
        BTFSS  FSR, 7     ; all done?
        GOTO   NEXT       ; no clear NEXT
CONTINUE                  ; yes CONTINUE
        NOP

; Set up prescaler
        banksel     OPTION_REG          ;Set prescaler to one (inc every microsec-
ond)
        movlw       B'00001000'
        movwf       OPTION_REG

init:
        banksel pulseCount              ;Clear pulseCount
        clrf    pulseCount
        banksel     pitCount            ;Clear pitCount
        clrf    pitCount

        movlw       .1                  ;Define pitID
        banksel     pitID
```

LED Register Code Development - 0.2

```
    movwf   pitID

    banksel ANSEL           ;Initialise digital output
    clrf    ANSEL
    banksel ANSELH
    clrf    ANSELH

    banksel PORTA           ;Clear outputs
    clrf    PORTA
    banksel PORTB
    clrf    PORTB
    banksel PORTC
    clrf    PORTC

    banksel TRISA           ;Initialise outputs
    clrf    TRISA
    banksel TRISB
    clrf    TRISB
    banksel TRISC
    clrf    TRISC

    movlw   B'00001000'     ;Initialise E3 as data input
    banksel TRISE
    movwf   TRISE

pulseIn:                    ;Listen on E3 for high input and measure length in
microseconds
    banksel PORTE
    btfss   PORTE,3         ;Is E3 high yet?
    goto    pulseIn         ;No, keep checking
    banksel TMR0            ;Yes, start timing
    clrf    TMR0
    banksel PORTE

timerLoop:
    btfsc   PORTE,3         ;Is E3 low yet?
    goto    timerLoop       ;No, keep timing
    banksel TMR0            ;Yes, store pulse length in pulseLength
    movfw   TMR0
    banksel pulseLength
    movwf   pulseLength

checkStartBit:              ;Check if pulse was a start bit (88us or higher) and
reset counters if so
    sublw .88               ;Subtract 88 from pulse length
    banksel STATUS
    btfsc   STATUS,0        ;If answer is +ve/0, pulse length was greater than 88
    goto    checkPulseCount ;Answer not +ve/0, therefore checkPulseCount
    banksel pitCount        ;Answer +ve/0, therefore pulse was a start bit - let
pitCount = 1
    clrf    pitCount
```

LED Register Code Development - 0.2

```
    incf   pitCount          ;Reset pulseCount
    banksel    pulseCount
    clrf   pulseCount
    goto   pulseIn

checkPulseCount:
    banksel    pulseCount
    movfw      pulseCount    ;Check if pulseCount = 24 and reset if so
    sublw .24
    banksel STATUS
    btfss  STATUS,2
    goto   switch            ;pulseCount != 24, switch
    banksel    pulseCount    ;pulseCount = 24, reset
    clrf   pulseCount
    banksel    pitCount
    incf   pitCount

switch:
    clrw                     ;Check pitCount not zero
    banksel    pitCount
    subwf pitCount
    banksel    STATUS
    btfsc  STATUS,2
    goto   pulseIn           ;pitCount zero, goto pulseIn, else...
    banksel pitID            ;Check if pitID = pitCount
    movfw      pitID
    subwf pitCount,W
    btfss  STATUS,2
    goto   incPulseCount     ;pitID != pitCount, therefore increment pulseCount and
continue listening
    banksel    pulseLength   ;pitID = pitCount, reload working register with
pulse length
    movfw      pulseLength
    sublw .63                ;Subtract 63 from pulse length
    banksel    STATUS
    btfsc  STATUS,0          ;If answer is +ve/0, pulse length was less than/equal to
63
    goto   switchOn          ;Therefore switch on
    goto   switchOff         ;Else, switch off

switchOn:
    call   setPort
    banksel    FSR
    movwf      FSR
    call   setBit
    banksel    INDF
    iorwf INDF,1
    banksel    pulseCount
    incf   pulseCount
    goto   pulseIn
```

LED Register Code Development - 0.2

```
switchOff:
    call   setPort
    banksel    FSR
    movwf      FSR
    call   setBit
    movwf      maskComp
    comf   maskComp
    movfw      maskComp
    banksel    INDF
    andwfINDF,1
    banksel    pulseCount
    incf   pulseCount
    goto   pulseIn

incPulseCount:              ;Increments pulseCount
    banksel    pulseCount
    incf   pulseCount
    goto   pulseIn

setPort:                    ;Port lookup table; returns PORT address for use with file
select register
    banksel    pulseCount
    movfw      pulseCount
    addwf      PCL
    retlw PORTC         ;1
    retlw PORTC         ;2
    retlw PORTC         ;3
    retlw PORTC         ;4
    retlw PORTC         ;5
    retlw PORTC         ;6
    retlw PORTC         ;7
    retlw PORTC         ;8
    retlw PORTB         ;9
    retlw PORTB         ;10
    retlw PORTA         ;11
    retlw PORTA         ;12
    retlw PORTB         ;13
    retlw PORTB         ;14
    retlw PORTA         ;15
    retlw PORTA         ;16
    retlw PORTB         ;17
    retlw PORTB         ;18
    retlw PORTA         ;19
    retlw PORTA         ;20
    retlw PORTB         ;21
    retlw PORTB         ;22
    retlw PORTA         ;23
    retlw PORTA         ;24

setBit:                     ;Bit lookup table; returns bitmask containing LED
location
```

LED Register Code Development - 0.2

```
banksel   pulseCount
movfw     pulseCount
addwfPCL
retlw B'00010000';1
retlw B'10000000';2
retlw B'00001000';3
retlw B'00000100';4
retlw B'01000000';5
retlw B'00100000';6
retlw B'00000010';7
retlw B'00000001';8
retlw B'00000010';9
retlw B'00000001';10
retlw B'01000000';11
retlw B'10000000';12
retlw B'00001000';13
retlw B'00000100';14
retlw B'00100000';15
retlw B'00010000';16
retlw B'00100000';17
retlw B'00010000';18
retlw B'00001000';19
retlw B'00000100';20
retlw B'10000000';21
retlw B'01000000';22
retlw B'00000010';23
retlw B'00000001';24

END
```

LED Register Code Development - 0.2

I used the same techniques as with register code 0.1 for testing during development (extensive use of MPLAB SIM's Watch and Stimulus panels)

Improvements

- Multiple register support (up to 255 registers on one line) (necessary addition)
- Processing speed improvements, to allow shorter delay between bits
- Refactored code to improve readability
- Added more commenting to improve readability

Register code 0.1 was very much a proof of concept for me, whereas 0.2 has everything necessary for use in the pit boards and as such is the version I will be programming into the registers once I have produced the boards.

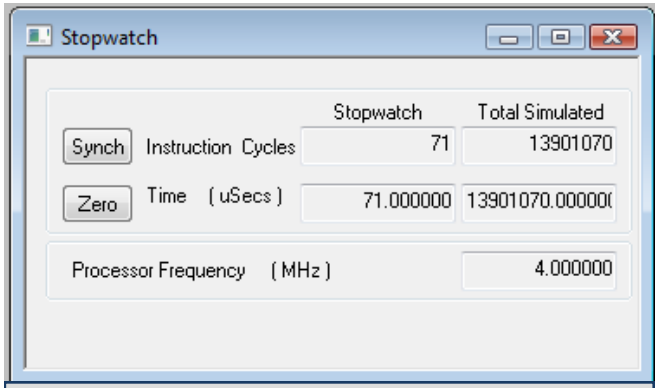


Figure 34.1 - Stopwatch

Speed Analysis

Out of curiosity, I did a series of tests to determine how long processing takes with this version. Using the Stopwatch panel (figure 34.1) of the MPLAB SIM debugging tool, I measured the time taken between the point at which the input line goes low (end of a bit) and the point at which the program returns to polling the input line, for a 'start bit', an 'on bit', and an 'off bit'. I set breakpoints at each of these points (figure 34.2) to allow me to zero the stopwatch/read the time from it.

Test Results

Start bit - 18µs
On bit - 71µs
Off bit - 75µs

```
182     timerLoop:
183         btfsc   PORTE,3         ;Is E3 low yet?
184         goto    timerLoop       ;No, keep timing
185     B   banksel TMR0            ;Yes, store pulse length in pulseLength
186         movfw   TMR0
187         banksel pulseLength
188         movwf   pulseLength
```

Figure 34.2 - Breakpoint on timerLoop

LED Register Code Development - 0.2

Evaluation

Without calculation it is clear that even with the processing improvements this version doesn't actually meet my specification for the control protocol, as it requires that the wait between bits is 25µs, while this code will require up to 75µs (and thus it would be a good idea to have the LED driver function always do this to prevent errors). There are two ways I could improve this:

1. Attempt to further optimise the register code. However I believe it is near optimal already so it is very unlikely all processing times could be reduced to 25µs.
2. Increase the clock speed of the PIC16F882s (and update the register code decision constants or prescaler) accordingly to 16MHz for example (this would reduce the maximum delay to 18.75µs).

However, when some basic calculations are done to see what the implications of the current processing times are, it becomes clear that they are actually of no consequence to the operation of the game.

To calculate the minimum update rate possible, divide one by the maximum packet length that can occur (which occurs when every LED is off). The length of this is:
1 x (100 + 18) [start bit] + 336 x (75 + 75) [off bit] = 50518µs = 50.518×10^{-3}s
Frequency then, equals $1/50.518 \times 10^{-6}$s = 19.7949Hz

An update rate of 20Hz is more than adequate for game play, since sowing should be visible to the user, and just about good enough for visual effects at the end of the game.

LED Register Code Development - 0.2

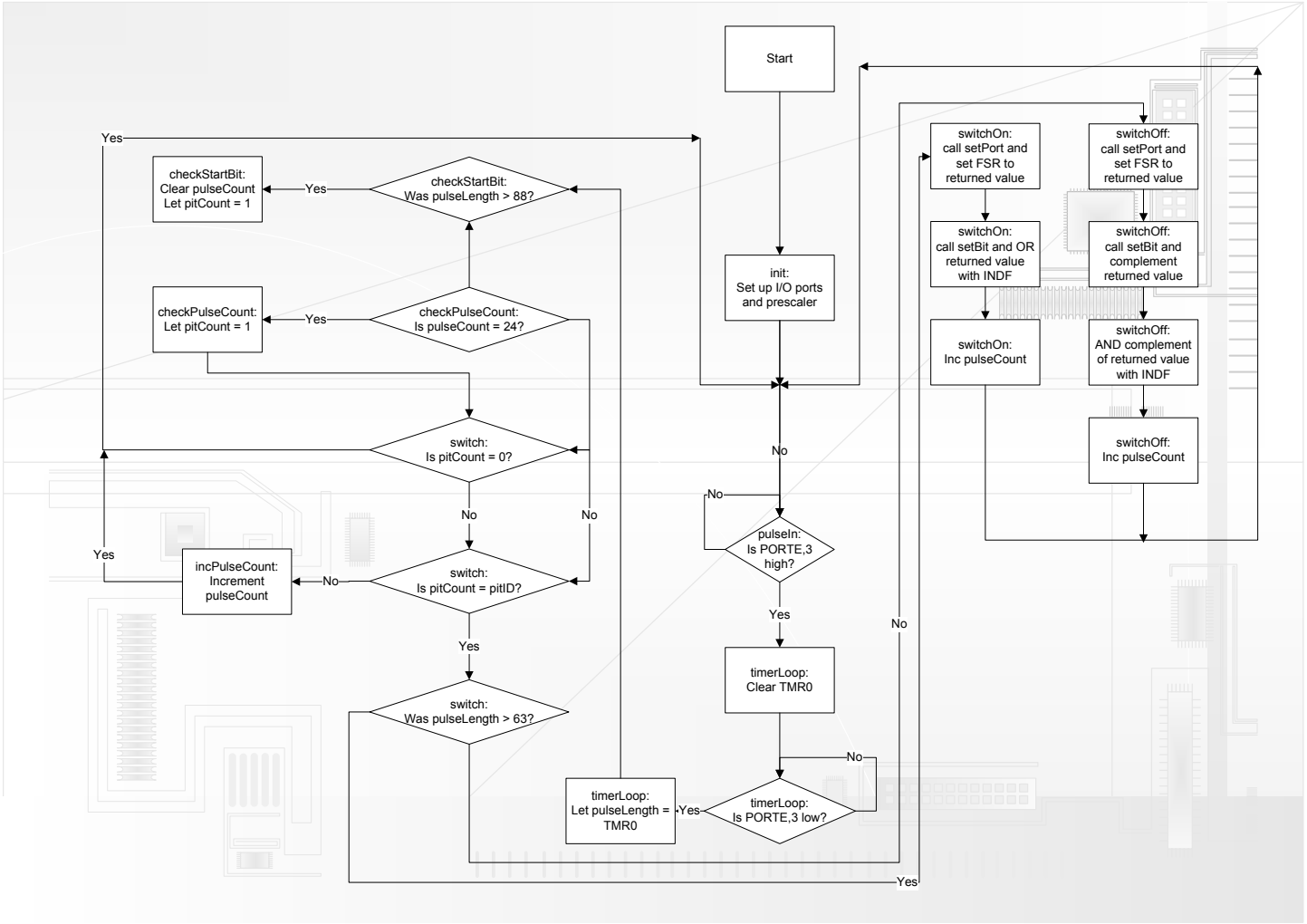


Figure 34.3 - Summary flowchart of register code 0.2

Mistakes in Artwork Set 0.2 - Player Two Upper

Figure 35.0 - Mistakes in Artwork Set 0.2 - Player Two Upper

I discovered these mistakes while drilling the component holes. The problem is missing tracks between the pads circled in red in figure 35.0, which I must have deleted in my haste to fix the pad size mistake and make up for the lost time. However, I decided it would not be worth the extra time to start the board again, so have worked around the problem by soldering wire links on the copper side.

PCB Pictures

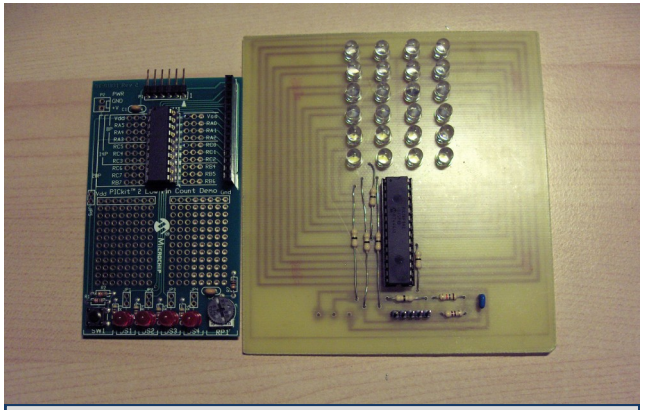


Figure 36.0 - PICKit 2 Demo Board and LED Register Prototype with Red, White and Blue LEDs for testing

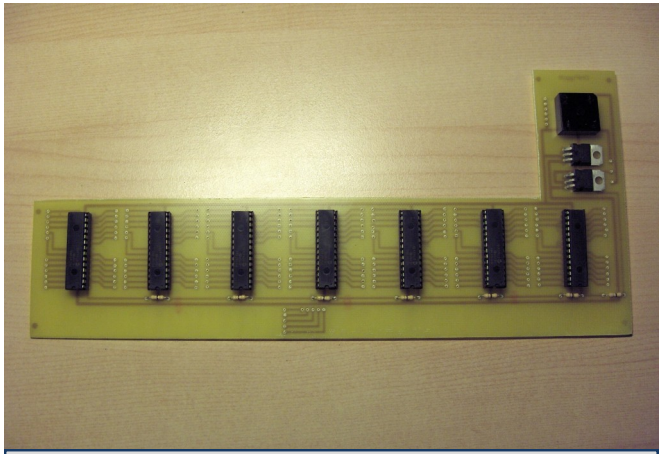


Figure 36.1 - Lower Pit Board (sans interconnects)

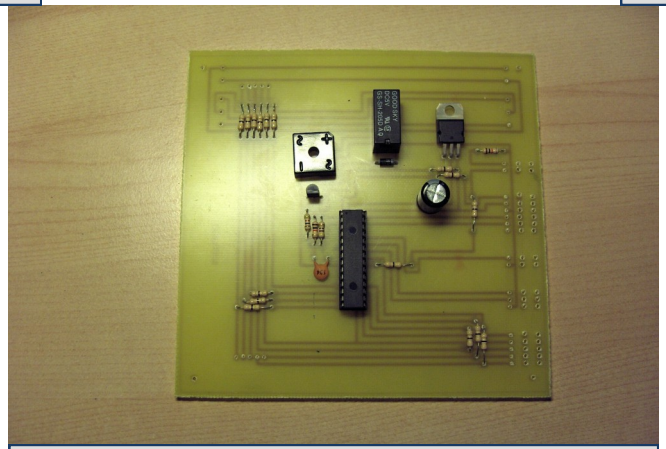


Figure 36.2 - Game Logic Board (sans interconnects) (artwork 0.2, PCB 0.1)

PCB Pictures

Figure 36.3 - Upper Pit Board prior to population

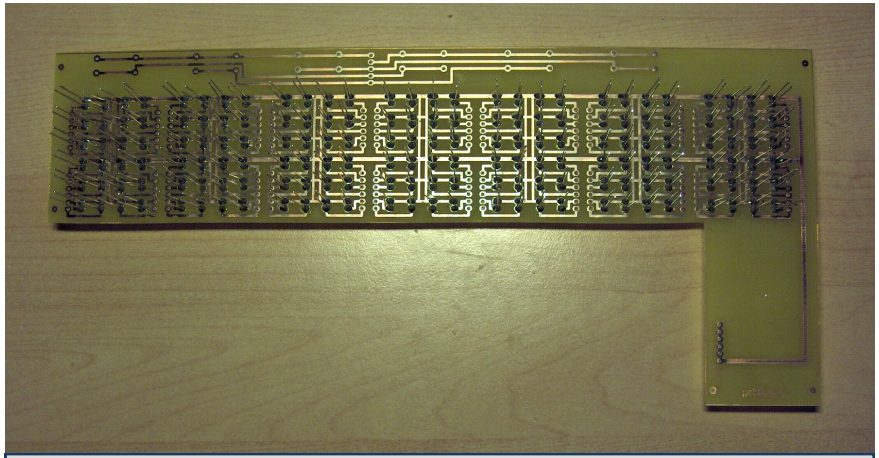


Figure 36.4 - Upper Pit Board, post population

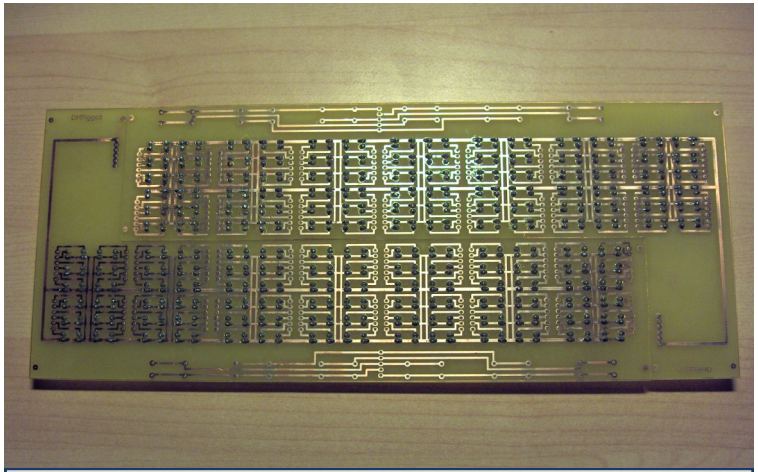


Figure 36.5 - Tessellated Upper Pit Boards (solder side, trimmed)

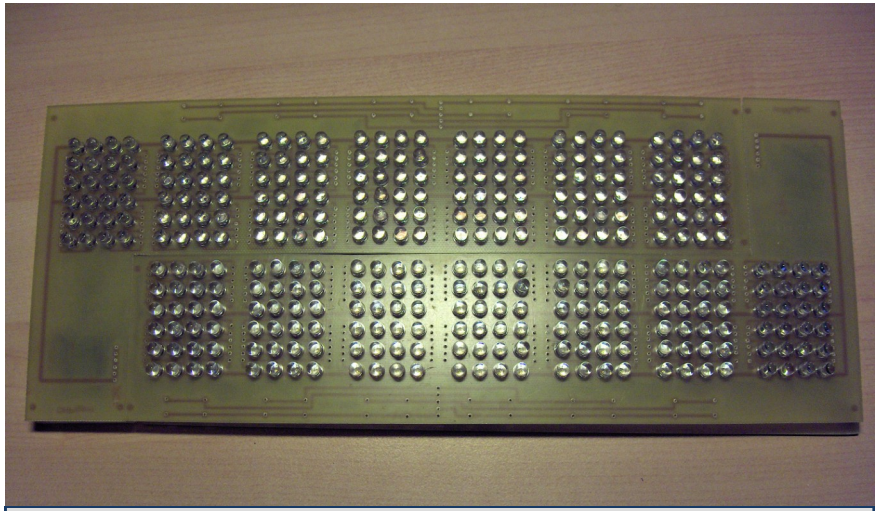


Figure 36.6 - Tessellated Upper Pit Boards (LED side)

PCB Pictures

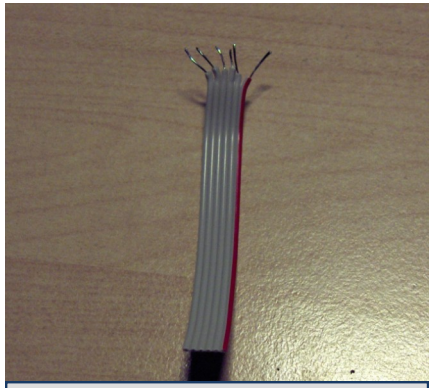


Figure 36.7 - One of 63 lengths of ribbon cable being prepared

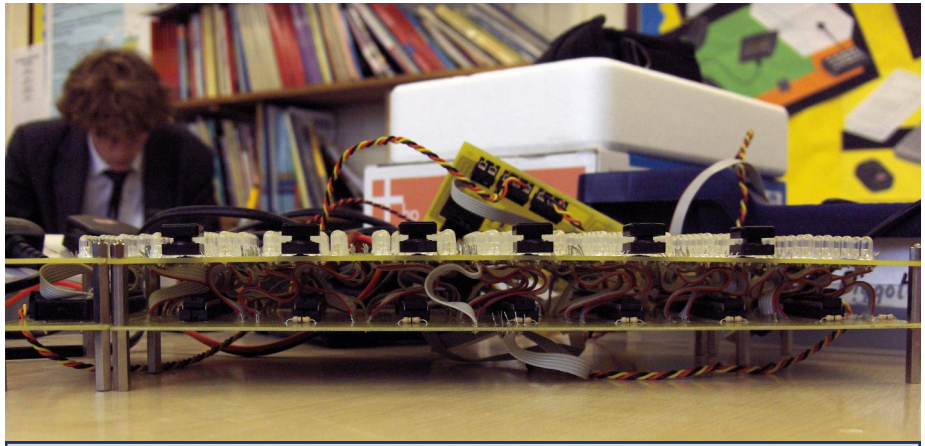


Figure 36.8 - Assembled pit boards with all interconnects

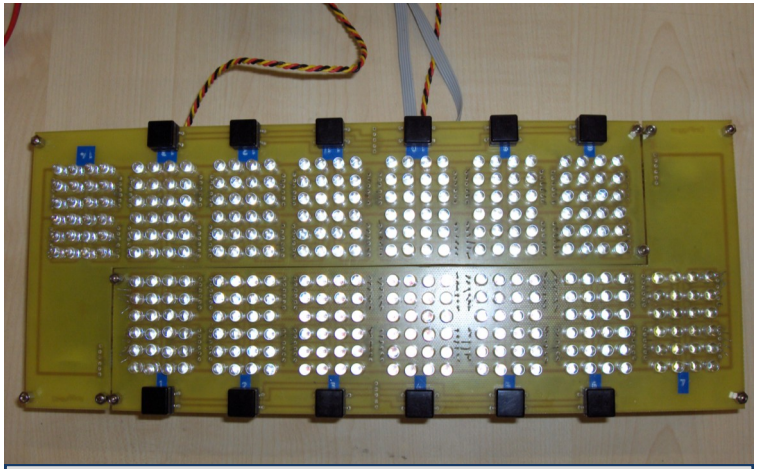


Figure 36.9 - Assembled and tessellated pit boards, with switches

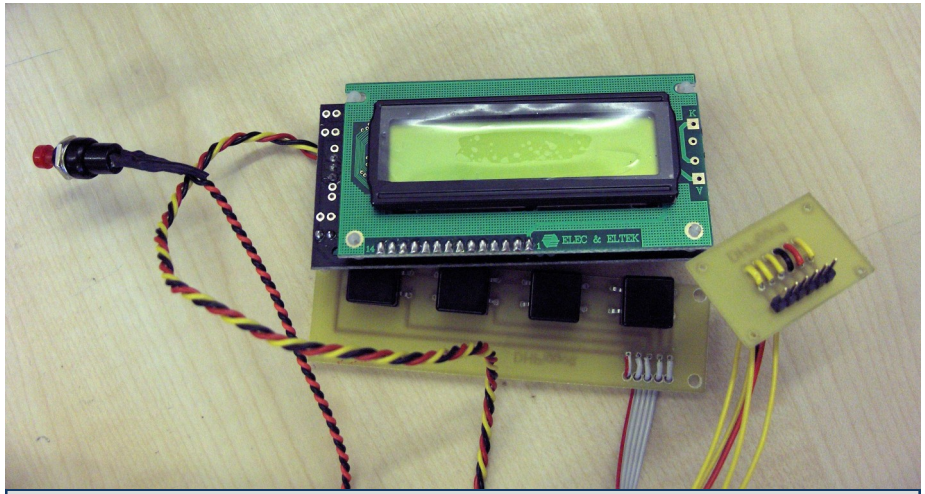


Figure 36.10 - Menu switches, LCD, ICSP socket and on/off button

Pit Boards Smoke Test

Having completed production of the pit boards, peripherals, and mainboard PCB 0.1 from artwork version 0.2, I was ready to test them. Although I would be learning C and using it for the actual game code, for the purposes of this smoke test, I chose to use assembly code because (a) I was already familiar with it, so was able to quickly write some test code and (b) The test code was relatively simple, so using such a low level language didn't slow me down.

I wrote the test code so that it would:

1) Switch on the LCD power
2) Switch on the power to the pit boards via the relay
3) Send a frame of data with the first and third column of each register on
4) Delay for approximately half a second
5) Send a frame of data with the second and fourth column of each register on
6) Delay for approximately half a second and then goto 3)

Because it was only test code I didn't spent any time on optimising memory usage, readability, or commenting it. For the sake of documentation completeness however, it is included in the pages that follow.

Figure 37.0 - Pit boards smoke test code

```
;*****************************************************************
;                                                                *
;   Filename:               18F252 Hardware Test                 *
;   File Version:           0.1                                  *
;                                                                *
;   Author:                 David Piggott                        *
;   Company:                piggott.me.uk                        *
;                                                                *
;*****************************************************************

        LIST P=18F252, F=INHX32         ;directive to define processor and file format
        #include <P18F252.INC>  ;processor specific variable definitions

;*****************************************************************************
******
;Configuration bits
;Microchip has changed the format for defining the configuration bits, please
;see the .inc file for futher details on notation.  Below are a few examples.



;   Oscillator Selection:
    CONFIG    OSC = XT          ;XT

;*****************************************************************************
******
;Variable definitions
; These variables are only needed if low priority interrupts are used.
; More variables may be needed to store other special function registers used
; in the interrupt routines.

                UDATA

WREG_TEMP       RES   1       ;variable in RAM for context saving
STATUS_TEMP     RES   1       ;variable in RAM for context saving
BSR_TEMP   RES   1       ;variable in RAM for context saving

                UDATA_ACS

EXAMPLE         RES   1       ;example of a variable in access RAM
myvar1          RES   1
myvar2          RES 1

;*****************************************************************************
******
;EEPROM data
; Data to be programmed into the Data EEPROM is defined here

DATA_EEPROM     CODE 0xf00000
```

Pit Boards Smoke Test

```
        DE    "Test Data",0,1,2,3,4,5
;*****************************************************************************
;******
;Reset vector
; This code will start executing when a reset occurs.
;
RESET_VECTOR    CODE 0x0000
        goto   init          ;go to start of main code
;*****************************************************************************
;******
;High priority interrupt vector
; This code will start executing when a high priority interrupt occurs or
; when any interrupt occurs if interrupt priorities are not enabled.
;
HI_INT_VECTOR   CODE 0x0008
        bra    HighInt       ;go to high priority interrupt routine
;*****************************************************************************
;******
;Low priority interrupt vector
; This code will start executing when a low priority interrupt occurs.
; This code can be removed if low priority interrupts are not used.
;
LOW_INT_VECTORCODE 0x0018
        bra    LowInt        ;go to low priority interrupt routine
;*****************************************************************************
;******
;High priority interrupt routine
; The high priority interrupt code is placed here.
;
        CODE
HighInt:
;       *** high priority interrupt code goes here ***
;
        retfie FAST
;*****************************************************************************
;******
;Low priority interrupt routine
; The low priority interrupt code is placed here.
; This code can be removed if low priority interrupts are not used.
```

Pit Boards Smoke Test

```
LowInt:
        movff STATUS,STATUS_TEMP   ;save STATUS register
        movff WREG,WREG_TEMP       ;save working register
        movff BSR,BSR_TEMP         ;save BSR register

;       *** low priority interrupt code goes here ***

        movff BSR_TEMP,BSR         ;restore BSR register
        movff WREG_TEMP,WREG       ;restore working register
        movff STATUS_TEMP,STATUS   ;restore STATUS register
        retfie

;*****************************************************************
;******
;Start of main program
; The main program code is placed here.
```

Pit Boards Smoke Test

```
init:
        clrf    LATA
        clrf    LATB
        clrf    LATC
        movlw   B'00000110'
        movwf   ADCON1
        clrf    TRISA
        clrf    TRISB
        clrf    TRISC
        movlw   B'00001111'
        movwf   TRISC
        movlw   B'00000010'
        movwf   TRISB
        bsf     LATB,5
        bsf     LATB,2

main:   call    processDelay
        call    startPulse
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1
        call    processDelay
        call    register1

        banksel myvar1
        clrf    myvar1
```

Pit Boards Smoke Test

```
delay1:
banksel     myvar2
clrf        myvar2
delay2:
banksel     myvar2
incf        myvar2
movlw       .255
subwf       myvar2
banksel     STATUS
btfss       STATUS,2
goto        delay2
banksel     myvar1
incf        myvar1
movlw       .255
subwf       myvar1
banksel     STATUS
btfss       STATUS,2
goto        delay1

call        processDelay
call        startPulse
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2
call        processDelay
call        register2

banksel     myvar1
```

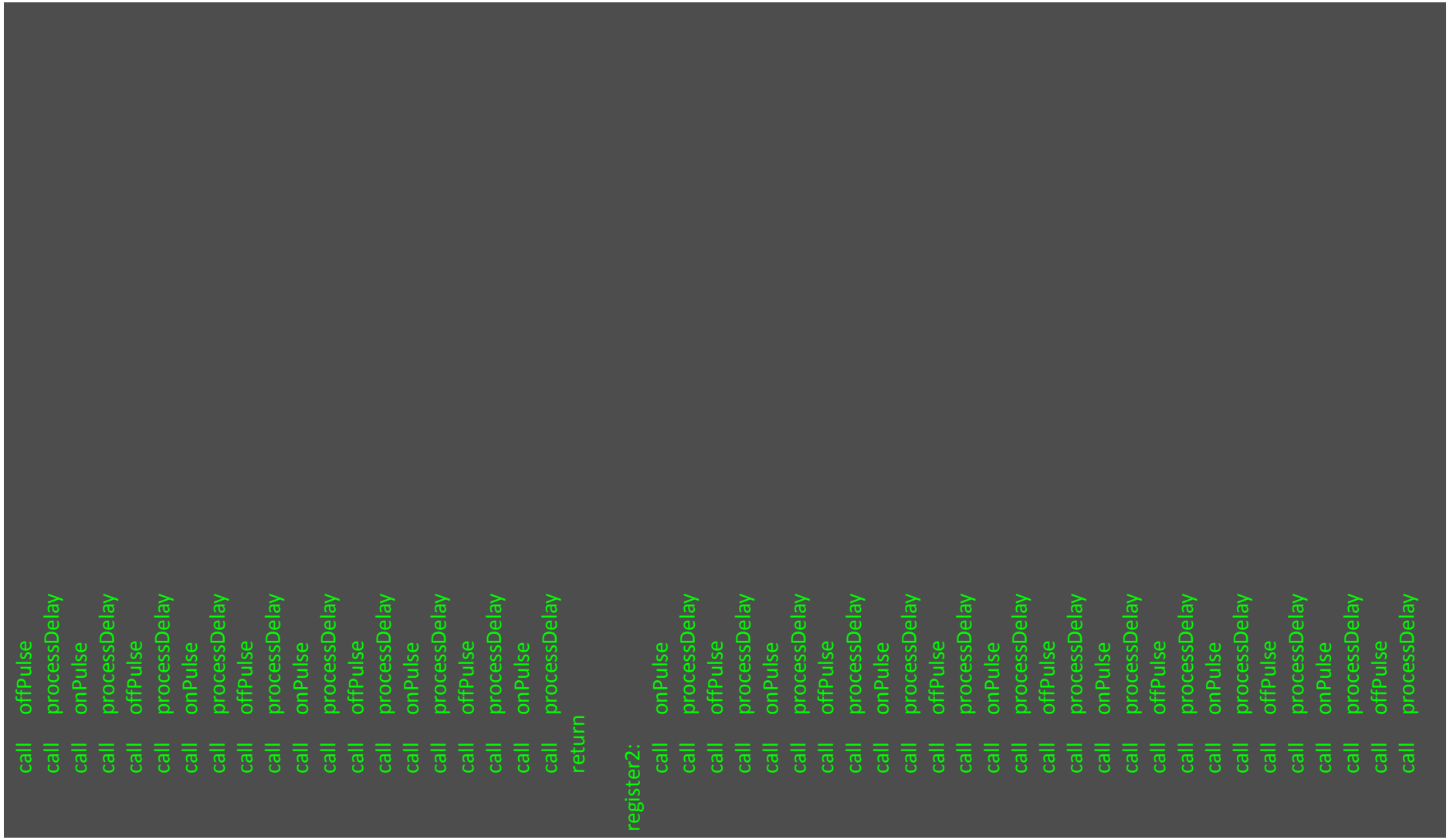
Pit Boards Smoke Test

```
    clrf    myvar1
delay3:
    banksel myvar2
    clrf    myvar2
delay4:
    banksel myvar2
    incf    myvar2
    movlw   .255
    subwf   myvar2
    banksel STATUS
    btfss   STATUS,2
    goto    delay4
    banksel myvar1
    incf    myvar1
    movlw   .255
    subwf   myvar1
    banksel STATUS
    btfss   STATUS,2
    goto    delay3

    goto  main

register1:
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
    call  offPulse
    call  processDelay
    call  onPulse
    call  processDelay
```

Pit Boards Smoke Test



Pit Boards Smoke Test

```
    call    onPulse
    call    processDelay
    call    offPulse
    call    processDelay
    call    onPulse
    call    processDelay
    call    offPulse
    call    processDelay
    call    onPulse
    call    processDelay
    call    offPulse
    call    processDelay
    call    onPulse
    call    processDelay
    call    offPulse
    call    processDelay
    call    onPulse
    call    processDelay
    call    offPulse
    call    processDelay
    return

processDelay:
    call    pause25us
    call    pause25us
    call    pause25us
    return

offPulse:
    banksel     PORTA
    bsf         PORTA,0
    call    pause25us
    call    pause25us
    call    pause25us
    bcf         PORTA,0
    return

onPulse:
    banksel     PORTA
    bsf         PORTA,0
    call    pause25us
    call    pause25us
    call    pause25us
    bcf         PORTA,0
    return

startPulse:
    banksel     PORTA
    bsf         PORTA,0
    call    pause25us
    call    pause25us
    call    pause25us
```

Pit Boards Smoke Test

```
        call    pause25us
        bcf             PORTA,0
        return

pause25us:
        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        return

;**********************************************************************
******
;End of program

                END
```

Pit Boards Smoke Test

This was the final version of the test code; I did have interim versions that progressed towards this (started with only one register, all LEDs on), but I think it better not to document all of these in my effort to keep the page count sensible.

I was able to download all of these test sequences successfully into the PIC18F252, but none of them would appear to run; that is , there was no click from the relay to even indicate power was getting to the pit boards. I tested the output of the pin controlling the relay and found it to be low. Knowing that it would probably be something really simple I was reluctant to open another support case with Microchip, but with no-where else to turn, did so anyway. However, while waiting for a reply, I had another very close look at the datasheet for the PIC18F252 and realised my mistake independently; I had foolishly assumed that because the PIC16F882s have built in 4MHz resonators that the same would be true for the PIC18F252. Not so. I would need to make another artwork and PCB with a resonator on.

However, still keen to know whether the pit boards themselves were functional, I checked the pin diagrams for the 18F252 and 16F882 and found that, for my circuit, they were pin compatible. I then ported the test sequence that I had written for the 18F252 so that it would run on the 16F882, swapped the PICs around, programmed the code to it, and was shortly pleased to hear the click of the relay. However, the LEDs on the pitboards remained decidedly unlit. I checked all the obvious things, that the register PICs had +5v across them, that the output of the driving PIC was pulsing, and that these pulses were also present on the input pins to the register PICs. All were as they should be. With no other leads to follow, I began to suspect corruption of the data frames, perhaps due to excessive capacitance (which I thought unlikely, given the low data rate). I hooked up a CRO to the input of one of the register PICs to check that the edges of the pulses were sharp. They were.

In the end this problem too was solved by re-examining the datasheet for the PIC (16F882 this time) very closely. On reaching the page about PORTE (that which the data input pin is on, on the register PICs), I noticed that the pin also doubles up as MCLR (master clear). I now realised that you must specify in the device configuration bits whether this should be used as reset or an input. On checking the device configuration bits in MPLAB IDE, I found that PORTE,3 was indeed configured as reset. If, as I suspected it to be, this was the cause of all pit boards not re-sponding, I would need to remove and reprogram each of the 14 register PICs. I first of all reprogrammed only the register PIC for pit one, and, pleased that this had solved the problem, continued to reprogram all the register PICs. Now all that remained to be done was make a new mainboard. Figure 37.1 overleaf shows this test sequence running.

Pit Boards Smoke Test

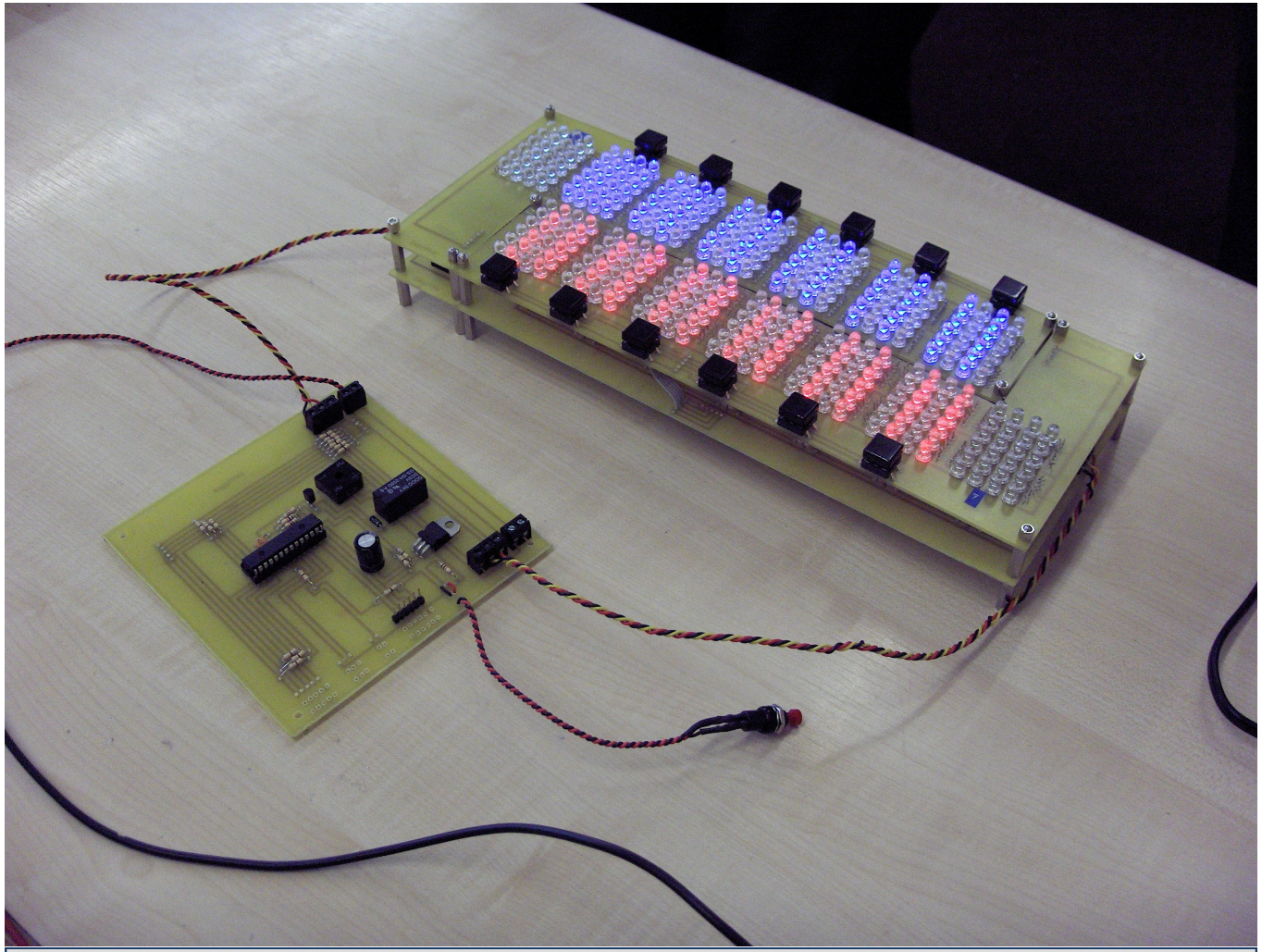


Figure 37.1 - Smoke test running on pit boards and main board 0.1 from artwork 0.2

Pit Boards Smoke Test

Power Issues

It should be noted that the boards would not power up until I raised the voltage of the power supply to 7.5V. Once they had powered up at this however I was able to reduce the power supply voltage down to 6V with everything still working fine. This is no doubt due to the voltage drop across the bridge rectifiers, and I do not consider this an issue in itself as I had planned to the run the boards from 12V input (9V upwards would be fine too but for high current applications, 12V supplies seem to be the standard).

Not visible in figure 37.1 is an issue with some of the red LEDs. This can be seen better in figure 37.2. I loaded a test program onto the board that switches on the first eight LEDs of the main pits. There are no problems with the blue LEDs at all. However, the characteristics of the problems with red are described below:

- The first seven red LEDs do not switch on at high board input voltages (despite the 5V regulators between power supply and PICs).
- The remaining seventeen LEDs operate with as much flexibility as the blue LEDs; i.e. given any board input voltage between 6V and 12V.
- The first seven can only be made to switch on by lowering the board input voltage to 6V at the supply. Once they have switch on they operate with board input voltages of 6V or 7.5V.
- Note however that, as described above, for the boards to power up, the board input voltage must be at least 7.5V, but for the red LEDs to subsequently light it must be lowered to 6V. It is acceptable to be having to do this while the game is still in development, but by the time it is finished, it should “just work”.

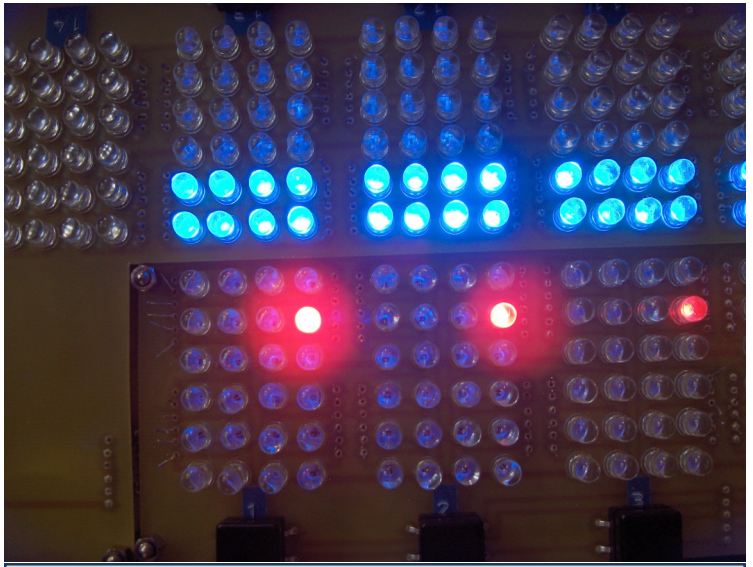


Figure 37.2 - Problem powering red LEDs

Pit Boards Smoke Test

In order to determine the cause of the problem, I took voltage measurements at various points on the (player one, red) board, with two different board input voltages:

Board input voltage; 6V

Pit board ground rail: 0.78V
Pit board +5V rail: 3.48
Output from PIC to a red LED: 2.61V
P.D. between pit board “ground” and “+5V” (i.e. powering the PIC16F882s): 2.7V
P.D. between PIC output and ground (i.e. powering the red LEDs): 1.83V

Board input voltage; 7.5V

Pit board ground rail: 0.94V
Pit board +5V rail: 4.89V
Output from PIC to a red LED: 2.65V
P.D. between pit board “ground” and “+5V” (i.e. powering the PIC16F882s): 3.95V
P.D. between PIC output and ground (i.e. powering the red LEDs): 1.71V

From figure 33.1 is can be seen that the problematic LEDs (the first seven) are all connected to PORTC of the register PICs. This has to be of some significance.

However, in the constant feud with time I delayed investigation of this problem in favour of completing the development and production of a final mainboard, which still had problems.

Oscillator Selection & Mainboard Artwork 0.3

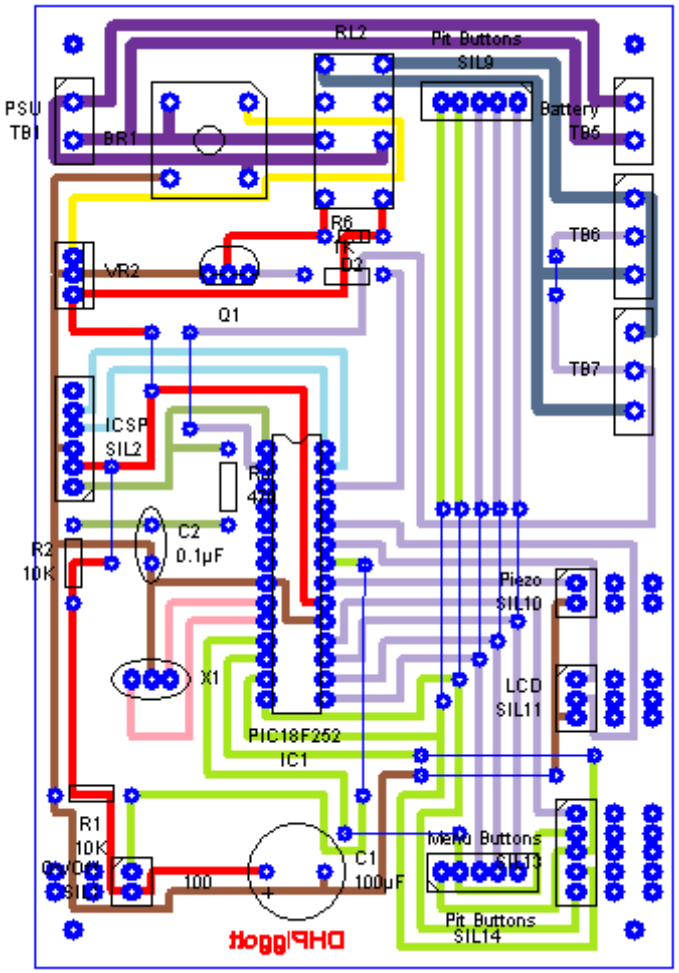


Figure 38.0 - Mainboard artwork 0.3

Figure 38.0 shows the new artwork, 0.3, that includes connections for a 3 pin ceramic resonator (X1). This is the only change between 0.2 and 0.3, however, inputs and outputs may also be tracked to different pins (these changes are detailed later on, with revised keyboard array addresses). Figure 40.1 shows the colour coding.

Another problem I had noticed was that I had incorrectly tracked the power button and its pull down resistor in artwork 0.2. This is fixed in artwork 0.3.

Figure 38.1 shows the oscillator modes available on the PIC18F252 (taken from page 19 of the PIC18F252 datasheet). In choosing what option to go for, I was able to eliminate the last four immediately, and so had to choose from the first four. The low power crystal runs too slowly, in the KHz range, so the choice was between a crystal or ceramic resonator, and whether to use PLL.

I decided to use a ceramic resonator because these are available from Rapid electronics in packages with the necessary capacitors built in, and this would simplify the artwork.

In preparation for the event that I finish all other tasks ahead of time and attempt to develop an AI player, I decided to run the 18F252 at the highest speed possible; 40Mhz. However the faster resonator available from Rapid was 32Mhz, so instead I chose to order a 10MHz resonator and use the device in PLL mode (multiplies resonator speed by four).

The PIC18FXX2 can be operated in eight different Oscillator modes. The user can program three configuration bits (FOSC2, FOSC1, and FOSC0) to select one of these eight modes:

1.	LP	Low Power Crystal
2.	XT	Crystal/Resonator
3.	HS	High Speed Crystal/Resonator
4.	HS + PLL	High Speed Crystal/Resonator with PLL enabled
5.	RC	External Resistor/Capacitor
6.	RCIO	External Resistor/Capacitor with I/O pin enabled
7.	EC	External Clock
8.	ECIO	External Clock with I/O pin enabled

Figure 38.1 - 18F252 Oscillator modes

Mainboard PCB 0.2 from artwork 0.3

Figure 39.1 shows the newly created mainboard 0.2. Having finished production of this board, I then set about updating the switch array addresses. Figure 39.0 shows the revised mappings.

	RC0	RC1	RC2	RC3
RC4	Pit 6	Pit 3	Pit 10	Pit 13
RC5	Pit 4	Pit 1	Pit 8	Pit 11
RC6	Pit 5	Pit 2	Pit 9	Pit 12
RB7	Menu 1	Menu 2	Menu 3	Menu 4

Figure 39.0 - Switch array addresses table

On doing so, I recognised a fault common to both this artwork and the previous; I had forgotten to include any pull up/down resistors for the switch array inputs. I would need to make yet another artwork and PCB from this.

Prior to doing this however, I put a PIC18F252 in the and programmed the test sequence I had written in assembly code onto it, so that I would at least know that the resonator was now working correctly. However when I populated the board, the 10MHz resonator I had ordered had not arrived (and still hasn't, as I write this documentation), so instead I had substituted it with a 4MHz resonator (which the department has a good supply of for use with PICAXEs). Accordingly, I set the oscillator configuration bit to XT (rather than HS + PLL which I will use with the 10MHz resonator).

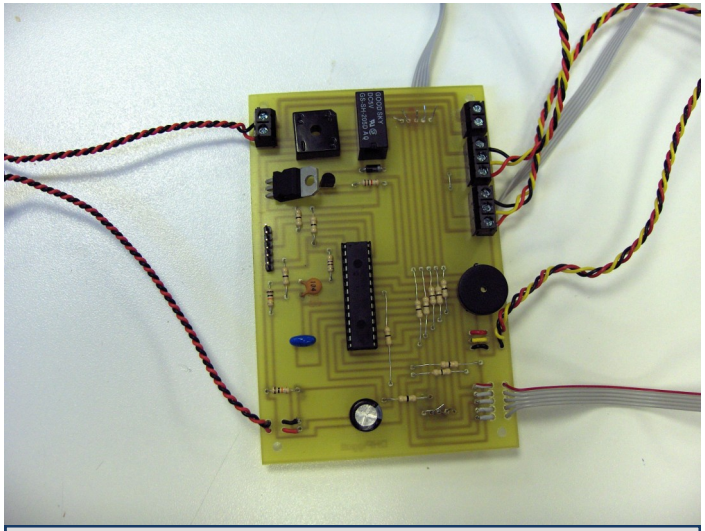


Figure 39.1 - Mainboard PCB 0.2 from artwork 0.3

I was relieved to find that the PIC18F252 was fully functional in this new board, and that my test sequence was correctly alternating the columns of LEDs on the pit boards.

Mainboard Artwork 0.4

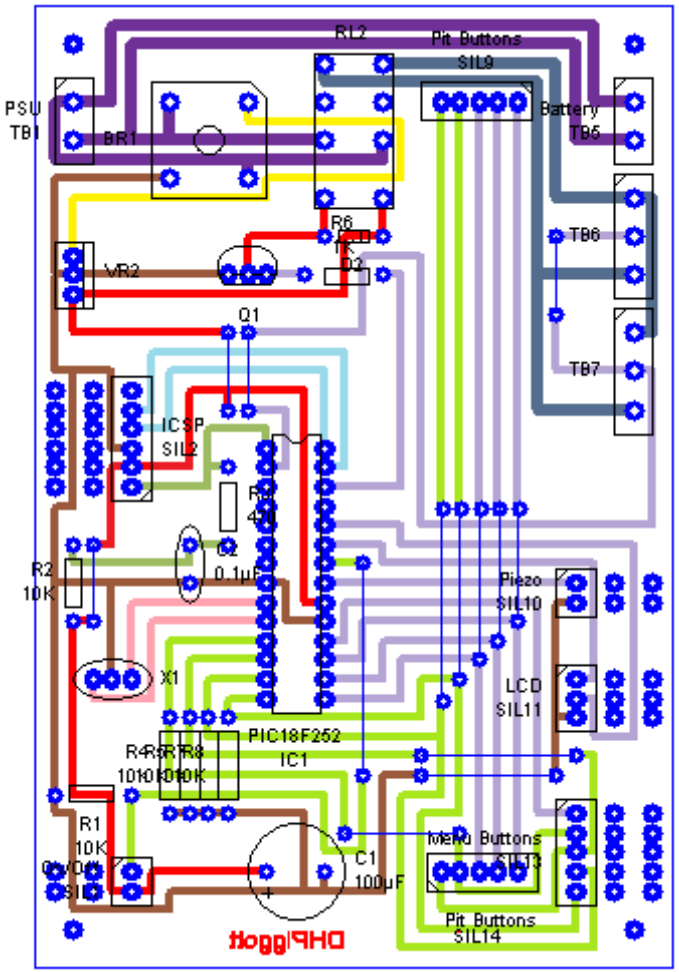


Figure 40.0 - Mainboard artwork 0.4

Figure 40.0 shows mainboard artwork 0.4. As mentioned, the only change of significance is that I have added four pull down resistors to ground, one for each of the inputs of the switch array.

Minor changes are that I have added weaving holes for the ICSP as I changed my mind again and decided to connect the header on the mainboard to the separate ICSP socket board I had made, for easy programming without opening the case. I had omitted these from artwork 0.3 as I had intended to position the PCB to one end of the pit boards, so the header would be adjacent to one edge of the case (and thus with a right angled header, it would be reachable via a hole in the side of the case). I repositioned a few pads to reduce/ increase the length of a few zero Ohm resistors, and generally tidied up the ICSP/resonator regions.

	Power Supply
	Relay Controlled Register Power Supply
	0v
	Unregulated +V
	+5v
	Input
	Output
	Programming clock and data lines
	V_{pp} Programming enable voltage
	Resonator

Figure 40.1 - Coding key

Mainboard PCB 0.3 from artwork 0.4

Figure 41.0 shows mainboard 0.3, completed with the exception of the piezo sounder. The ‘smoke test’ code ran successfully on this too as expected. The switch array addresses are identical to those of mainboard 0.2 from artwork 0.3 (figure 39.0).

This concludes all hardware development. The significant remaining tasks are:

1) Game code development
2) Case design
3) LCD & menu code development
4) Case construction
5) AI player development (time and PIC18F252 capabilities permitting)

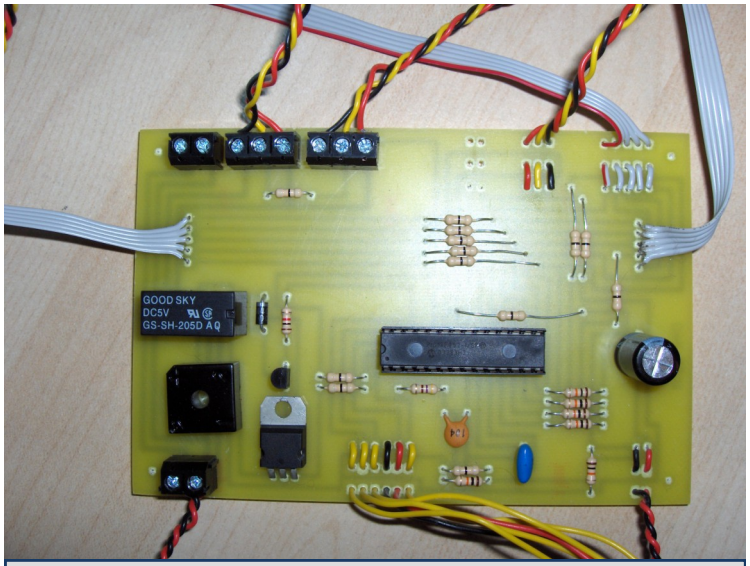


Figure 41.0 - Mainboard PCB 0.3 from artwork 0.4

Production Time Plan

ID	Task Name	Start	Finish	Duration
1	Christmas Holidays	15/12/2007	06/01/2008	23d
2	Spring Half Term	09/02/2008	17/02/2008	9d
3	Easter Holidays	20/03/2008	06/04/2008	18d
4	Project initialisation	01/11/2007	09/11/2007	9d
5	Development	11/11/2007	25/02/2008	107d
6	Case dimensioning & model creation	26/02/2008	07/03/2008	11d
7	Upper and lower panel production	10/03/2008	14/03/2008	5d
8	Side panel production	17/03/2008	19/03/2008	3d
9	Completion of case (spirowrap, rubber feet, battery boxes etc.)	17/03/2008	19/03/2008	3d
10	LCD menus & AI code development	20/03/2008	06/04/2008	18d
11	Completion of documentation	07/04/2008	10/04/2008	4d

Figure 42.0 - Time management review

Time Management Review

Figure 42.0 shows my actual time use of time to the present day. As can be seen from the tasks shown in it, I have infact at this stage already introduced myself to the C programming language and written a proof of concept program; that is, a basic, but playable game of iBantumi!

Also worth noting at this time is that the deadline for A2 coursework has been altered, from 19/03/08 to 10/04/08, for both practical and documentation.

This is brilliant news for the project because I will no longer have to create the case and code all the peripheral features (LCD menus and AI player) in the limited time before the Easter holidays. Realistically I would probably have just had to drop plans for the AI player. I will now be able to use the time before the Easter holidays to complete the case, and then spend the Easter holidays finalising the main game logic program and the documentation.

Proof of Concept Program

Included here for the sake of completeness is the first program I wrote in C, that is a proof of concept for the iBantumi game. The following points should be noted:

- It is not fully commented, as I will be refactoring it when I start work on the final program.
- It is probably not the most efficient means to an end, but as stated is just a proof of concept so I did not spend a great deal of time on it.
- I will not explain at this point in documentation how it works; this will come when I have completed case development and construction and start work on the final program.
- It is not complete; there is no evaluation of the game state to determine if a player has won, so the game doesn't terminate as might be expected.
- Though I did extensive play testing, and there was no shortage of volunteers for this (it has already proved to be a highly enjoyable game!), there are still one or two "bugs". I say that in quotation marks because they do not cause the program to crash, merely misapply one or two rules of the game.
- At this stage the resonator in the mainboard is still a 4MHz device, so delays are coded accordingly.

Features of this program:

- Player one (red) always starts the game.
- There is a brief delay between each seed that is sown to allow players to watch moves take place.
- As I have not yet worked out how to control the LCD, there are no menus and instead the four menu buttons are used to restart the game; one button starts the game with one seed per pit, the next button with two per pit, the next with three and the last with four.

Proof of Concept Program

Figure 43.0 - Proof of concept program

```
#include <p18cxxx.h>
#pragma config WDT = OFF

//          Function prototypes
int         scankeys();
int         kevin();
void  writeLEDs();
void  delay25us();
void  updateDelay();

//          Variable declarations
int         key;
int         pitData[15];
int         LEDDriverPitCount;
int         activePit;
int         activePlayer;
long  delayCount;

//          Function declarations
int         scankeys(){
      LATC = 0b00010000;

      switch(0b00001111 & PORTC){
      case 0b00000001:
            key = 6;
      break;
      case 0b00000010:
            key = 3;
      break;
      case 0b00000100:
            key = 10;
      break;
      case 0b00001000:
            key = 13;
      break;
      default:
            key = 0;
      break;
      }

      if(key > 0){
            return      key;
      }

      LATC = 0b00100000;
      switch(0b00001111 & PORTC){
      case 0b00000001:
            key = 4;
      break;
```

Proof of Concept Program

```
case 0b00000010:
        key = 1;
    break;
    case 0b00000100:
        key = 8;
    break;
    case 0b00001000:
        key = 11;
    break;
    default:
        key = 0;
    break;
    }

    if(key > 0){
        return      key;
        }

    LATC = 0b01000000;
    switch(0b00001111 & PORTC){
    case 0b00000001:
        key = 5;
    break;
    case 0b00000010:
        key = 2;
    break;
    case 0b00000100:
        key = 9;
    break;
    case 0b00001000:
        key = 12;
    break;
    default:
        key = 0;
    break;
    }

    if(key > 0){
        return      key;
        }

    LATC = 0b10000000;
    switch(0b00001111 & PORTC){
    case 0b00000001:
        key = 15;
    break;
    case 0b00000010:
        key = 16;
    break;
    case 0b00000100:
        key = 17;
```

Proof of Concept Program

```
                break;
                case 0b00001000:
                        key = 18;
                break;
                default:
                        key = 0;
                break;
                }

                if(key > 0){
                        return key;
                }else{
                        return 0;
                }
        }

int     keyin(){
        key = 0;
        while(key == 0){
                key = scankeys();
                if(key != 0){
                        return key;
                }
        }
}

void  writeLEDs(){
        PORTAbits.RA0 = 1;
        delay25us();
        delay25us();
        delay25us();
        delay25us();
        delay25us();
        PORTAbits.RA0 = 0;
        delay25us();

        for(LEDDriverPitCount = 1; LEDDriverPitCount <= 14; LEDDriverPitCount++){

                int LED;

                switch(LEDDriverPitCount){
                        case 7:
                                for(LED = 0; LED < (24 - pitData[LEDDriverPitCount]); LED++){
                                        PORTAbits.AN0 = 1;
                                        delay25us();
                                        delay25us();
                                        delay25us();
                                        PORTAbits.RA0 = 0;
                                        delay25us();
                                }
                                for(LED = 0; LED < pitData[LEDDriverPitCount]; LED++){
                                        PORTAbits.AN0 = 1;
```

Proof of Concept Program

```
            delay25us();
            delay25us();
            PORTAbits.RA0 = 0;
            delay25us();
        }
    break;
    case 14:
        for(LED = 0; LED < (24 - pitData[LEDDriverPitCount]); LED++){
            PORTAbits.AN0 = 1;
            delay25us();
            delay25us();
            delay25us();
            PORTAbits.RA0 = 0;
            delay25us();
        }
        for(LED = 0; LED < pitData[LEDDriverPitCount]; LED++){
            PORTAbits.AN0 = 1;
            delay25us();
            delay25us();
            delay25us();
            PORTAbits.RA0 = 0;
            delay25us();
        }
    break;
    default:
        for(LED = 0; LED < (24 - pitData[LEDDriverPitCount]); LED++){
            PORTAbits.AN0 = 1;
            delay25us();
            delay25us();
            delay25us();
            PORTAbits.RA0 = 0;
            delay25us();
        }
        for(LED = 0; LED < pitData[LEDDriverPitCount]; LED++){
            PORTAbits.AN0 = 1;
            delay25us();
            delay25us();
            delay25us();
            PORTAbits.RA0 = 0;
            delay25us();
        }
    break;
        }
    }
}
void  delay25us(){
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
```

Proof of Concept Program

```
    Nop();
    Nop();
    Nop();
    Nop();
    Nop();

    Nop();
    Nop();
    Nop();
    Nop();
    Nop();

    Nop();
    Nop();
    Nop();
    Nop();
    Nop();

    Nop();
    Nop();
    Nop();
    Nop();
    Nop();
}

void  updateDelay(){
    for(delayCount = 0; delayCount < 10000; delayCount++){
    }
}

int     move(){
    if(pitData[0] != 0){
        activePit++;
        if(activePit == 15){
            activePit = 1;
        }else if(activePit == 7 && activePlayer == 1){
            activePit++;
        }else if(activePit == 14 && activePlayer == 0){
            activePit++;
        }
        pitData[activePit]++;
        pitData[0]--;
        return 1;
    }else{
        return 0;
    }
}

void  main(void){
    //      Initialise I/O
    LATA = 0;
```

Proof of Concept Program

```
LATB = 0;
LATB = 0;
ADCON1 = 0b00000110;
TRISA = 0b00000000;
TRISB = 0b00000010;
TRISC = 0b00001111;

//          Enable pitBoards power control relay and power to LCD
PORTBbits.RB2 = 1;      //      LCD
PORTBbits.RB5 = 1;      //      Relay

//          Initialise board with two seeds per pit, excluding mancala
for(LEDDriverPitCount = 1; LEDDriverPitCount < 15; LEDDriverPitCount++){
        pitData[LEDDriverPitCount] = 2;
    }
pitData[7] = 0;
pitData[14] = 0;

//          Send starting seeds to board
for(delayCount = 0; delayCount < 10; delayCount++){
        writeLEDs();
    }

activePlayer = 0;

while(1){
        key = keyin();
        //              Wait for player to initialise their move
        if((1 <= key && key <= 6 && activePlayer == 0) | (8 <= key && key <= 13 &&
activePlayer == 1)){        //      Perform one move
                pitData[0] = pitData[key];
                pitData[key] = 0;
                activePit = key;
                while(move()){
                        updateDelay();
                        for(delayCount = 0; delayCount < 10; delayCount++){
                                writeLEDs();
                        }
                }
                //      Move complete, evaluate stop conditions
                if((pitData[activePit] == 1) && (activePlayer == 0) && (1 <= ac-
tivePit && activePit <= 6) && (pitData[14 - activePit] != 0)){
                        pitData[7] += pitData[14 - activePit];
                        pitData[14 - activePit] = 0;
                        updateDelay();
                        for(delayCount = 0; delayCount < 10; delayCount++){
                                writeLEDs();
                        }
                        pitData[7] += pitData[activePit];
                        pitData[activePit] = 0;
```

Proof of Concept Program

```
                updateDelay();
                for(delayCount = 0; delayCount < 10; delayCount++){
                    writeLEDs();
                }
                activePlayer = 1 - activePlayer;
            }else if((pitData[activePit] == 1) && (activePlayer == 1) && (8 <=
activePit && activePit <= 13) && (pitData[14 - activePit] != 0)){
                pitData[14] += pitData[14 - activePit];
                pitData[14 - activePit] = 0;
                updateDelay();
                for(delayCount = 0; delayCount < 10; delayCount++){
                    writeLEDs();
                }
                pitData[14] += pitData[activePit];
                pitData[activePit] = 0;
                updateDelay();
                for(delayCount = 0; delayCount < 10; delayCount++){
                    writeLEDs();
                }
                activePlayer = 1 - activePlayer;
            }else if((activePit == 7 && activePlayer == 0) | (activePit == 14
&& activePlayer == 1)){
                }else{
                    activePlayer = 1 - activePlayer;
                }
            }else if(key == 15){
                //          Reset
                //          Initialise board with two seeds per pit, excluding mancala
                for(LEDDriverPitCount = 1; LEDDriverPitCount < 15; LEDDriverPit-
Count++){
                    pitData[LEDDriverPitCount] = 1;
                }
                pitData[7] = 0;
                pitData[14] = 0;

                //          Send starting seeds to board
                for(delayCount = 0; delayCount < 10; delayCount++){
                    writeLEDs();
                }
                activePlayer = 0;
            }else if(key == 16){
                //          Reset
                //          Initialise board with two seeds per pit, excluding mancala
                for(LEDDriverPitCount = 1; LEDDriverPitCount < 15; LEDDriverPit-
Count++){
                    pitData[LEDDriverPitCount] = 2;
                }
                pitData[7] = 0;
                pitData[14] = 0;

                //          Send starting seeds to board
```

Proof of Concept Program

```
        for(delayCount = 0; delayCount < 10; delayCount++){
            writeLEDs();
        }
        activePlayer = 0;
    }else if(key == 17){
                Reset
        //      Initialise board with two seeds per pit, excluding mancala
        for(LEDDriverPitCount = 1; LEDDriverPitCount < 15; LEDDriverPit-
Count++){
            pitData[LEDDriverPitCount] = 3;
        }
        pitData[7] = 0;
        pitData[14] = 0;

        //      Send starting seeds to board
        for(delayCount = 0; delayCount < 10; delayCount++){
            writeLEDs();
        }
        activePlayer = 0;
    }else if(key == 18){
        //      Reset
        //      Initialise board with four seeds per pit, excluding mancala
        for(LEDDriverPitCount = 1; LEDDriverPitCount < 15; LEDDriverPit-
Count++){
            pitData[LEDDriverPitCount] = 4;
        }
        pitData[7] = 0;
        pitData[14] = 0;

        //      Send starting seeds to board
        for(delayCount = 0; delayCount < 10; delayCount++){
            writeLEDs();
        }
        activePlayer = 0;
    }
}
```

Component Research - On/off button

Vandal resistant switches

Stainless steel, vandal resistant switches for applications in public environments where standard switches would be frequently damaged or vandalised.

- Typically used for door entry systems, elevators, vending machines etc.
- Three types of switch profile – flat, domed or prominent
- **IP66** sealed and **IP68** sealed available

Price: £3.65 for cheapest

As the power management is implemented entirely in software*, I could just as easily choose a momentary action switch in place of a toggle action switch. With this in mind, I looked at all the switch types available from Rapid, as there are essentially no criteria that the switch must match other than good aesthetics. The vandal resistant switch pictured opposite meets this.

There are numerous types of vandal resistant switch available (all momentary action) so I simply chose the cheapest, at £3.65 (prices range up to £8.50).

* Game logic PIC determines if game should be on/off and then switches LCD (powered by PIC output) and pit boards (via relay) accordingly, using sleep mode to conserve power so that in standby the theoretical current drawn is less than 2µA.

Component Research - Power Supply

30-60W Mini desktop switch mode PSU

A range of high quality switch mode PSUs that have 3-pin IEC input connectors.

- Wide range input voltage 90 to 264V AC at 47 to 63Hz
- Short circuit, over voltage and over current protection
- Typically 80% efficient
- Fully regulated output
- Connector, 2.1 x 5.5 x 12mm female barrel
- Dimensions 120 x 60 x 36mm
- IEC, TUV, UL approved

Price: £17.75 for 12V 5A version

When choosing a power supply, it is important that the supply is capable of providing enough current for the game to operate at full capacity; that is, with all LEDs on. This way I will be able to make use of the pitboards for special effects at certain points in play (such as when the game is over, to indicate the winner).

The maximum theoretical current drawn by the device is 25mA for 336 LEDs. Using an approximation of 2V for the p.d. across the LEDs, this is $25 \times 10^{-3} \times 2 \times 336 = 16.8$W. Power used by the microcontrollers/LCD is negligible.

Assuming an arbitrary 80% regulator efficiency, this means a current of 1.75A ((16.8/80 x 100) / 12) on the 12V side of the regulators, so the 30-60W Mini desktop switch mode PSU opposite should be more than adequate. The reason I have chosen this one rather than a lower power one is that I will not actually be ordering a power supply at all, in an effort to prevent further expenditure; this is the power supply used in my AS Heavy Sleeper's Alarm Clock, so I shall simply use that (and no doubt purchase one myself later so I can use both the alarm clock and iBantumi simultaneously).

In normal gameplay, the power used by the pitboards should only be $25 \times 10^{-3} \times 2 \times 24 = 1.2$W. This is a current of 0.125A ((1.2/80 x 100) / 12) on the 12V side of the regulators.

From the Wikipedia article on AA batteries (http://en.wikipedia.org/wiki/AA_battery):

> Alkaline batteries from 1700 mAh to almost 3000 mAh cost a little more, but last proportionally longer [than other battery chemistries].

There will be eight AA cells in series, so the estimate runtime for the game on battery power is (assuming 2.35Ah average capacity) 2.35/0.125 = 18.8 hours of normal gameplay!

Component Research - Power Socket

Heavy duty DC power socket

High quality DC power sockets featuring nickel plated bodies and single hole fixing.

- Suitable for panel thickness up to 2.5mm
- Panel cut-out 11mm
- Supplied with hexagonal fixing nut

Price: £0.55

I had intended to use the power socket pictured opposite, but at the time of ordering they were out of stock, and with a looming deadline the most sensible option was to substitute it with the power socket shown below, despite the significantly higher cost.

DC Power sockets

A Professional grade panel mounting DC power sockets in 2.1 and 2.5mm sizes, suitable with Switchcraft power plugs 20-0882 to 20-0892.

- Panel mounting
- Maximum panel thickness is 3.1mm
- Switched
- Contacts rated at 5A 12V DC
- Switchcraft types 722A and 712A

Price: £2.52

Component Research - Piezo & Resonator

3-pin Ceramic resonators (ACT)

A range of 3-pin ceramic resonators supplied with built-in capacitors.

- No need for load capacitance when making up oscillator circuits
- Leads on 2.5mm pitch spacing

Price: £0.25 for 10MHz version

As already discussed with the development of artwork 0.3, an external resonator is required for the main game logic PIC. In advance preparation for the event that I do have time for, and succeed in developing an AI player, it will be beneficial to be running the PIC at maximum speed to reduce processing times.

The maximum rated speed of the PIC18F252 is 40MHz, but the highest speed ceramic resonator available from Rapid is 32MHz, so instead I have chosen a 10MHz resonator and will configure the oscillator module in PLL (phase locked loop) mode for an effective speed of 40MHz.

While waiting for the resonator to arrive, I have been using a 4MHz resonator as school stock these for 28 and 40 pin PICAXEs (hence the delay timing in the proof of concept program being as it is).

The piezo resonator will be used to provide audible feedback during play and navigation of the menus.

When choosing a piezo transducer I was looking for three things;

1. High loudness.
2. On flying leads (i.e. not PCB mounting).
3. No built in drive circuit (the PIC18F252 will drive it).

Miniature piezo transducer

A miniature flange mounting piezo audio transducer with flying leads.

- Housed in a compact package
- Requires an external drive circuit
- Dimensions 29.5x5.2mm
- Soundtech type SEP-1126

Price: £0.56

Operating voltage (Vp-p max)	30V
Sound output	95dB at 10cm
Resonant frequency	2.8kHz
Current consumption	8mA
Capacitance (±30%)	18,000pF
Weight	4g
Operating temperature	–20°C to +60°C

Case Development

One of the specification points was that the case will be based on initial case idea six. This is shown in figure 44.0 as a reminder.

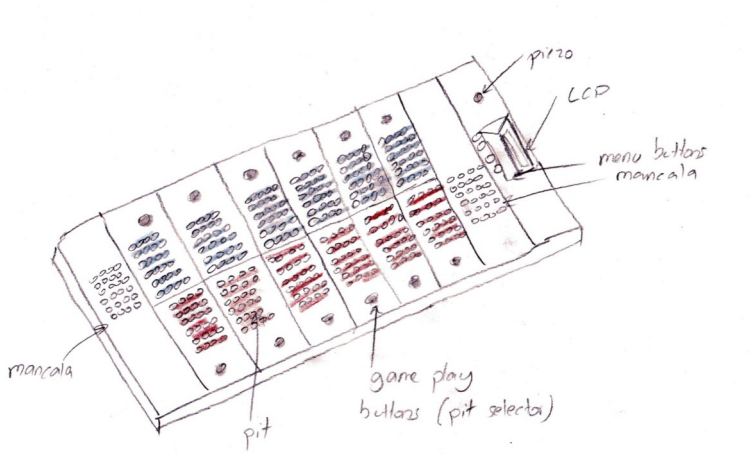


Figure 44.0 - Case development starting point

In order to develop my ideas further, I set about producing an exact model of every component that will be in the case, using PTC ProDESKTOP. This allowed me to make geometrically accurate designs and photorealistic renders of my ideas.

Figure 44.1 shows the initial shell idea I came up with; it's a two part structure consisting of an upper and lower, as shown in figure 44.2.

Unfortunately it would have been impractical to attempt production of this with the available tools and equipment; the pieces are too big for CAMM so the holes would have to be cut by hand which would be inaccurate. Additionally, creating the wide bends with the line bender would amplify any inaccuracies in the bending process and so result in a poor fit.

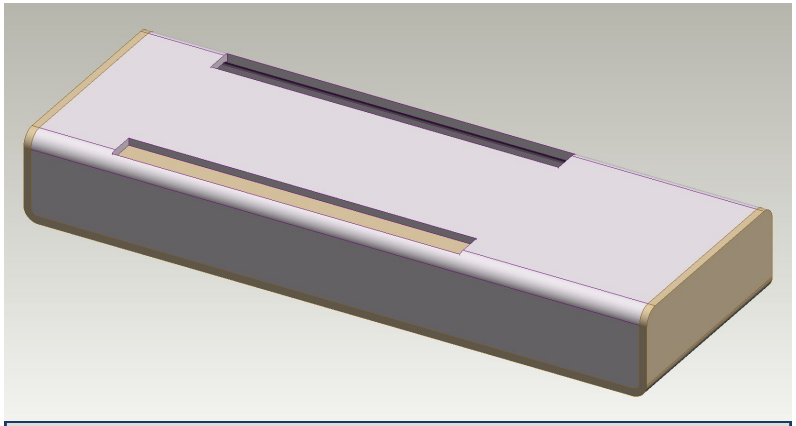


Figure 44.1 - Initial shell idea

For these reasons it was necessary to create an alternative case design that a) doesn't require wide bends and b) doesn't require large internal cut outs (due to it being too large for CAMM).

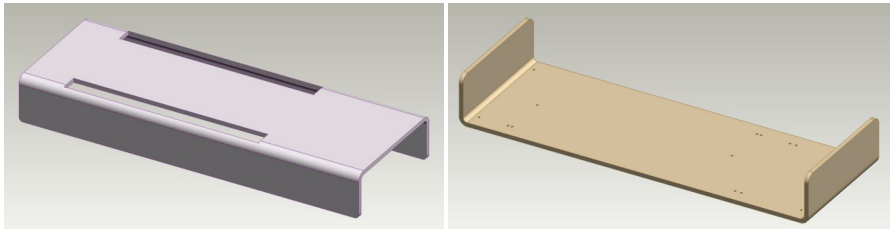


Figure 44.2 - Initial shell idea sections

Case Development

Component Creation

As already stated, I created exact models of every component that will be in the case. It is important that I make the meaning of the word 'component' in this context clear; I made a new component for every different material encountered. For example, when creating resistors, I didn't make the entire resistor in one design file; I broke it down into; 1) the ceramic body 2) the pins either side 3) the colour bands (one design for each colour of band, of each size (depends on whether the band is at the centre or end of the body). The reasons for doing this are twofold. Firstly, it generally makes the modelling process quicker, easier and keeps designs more organised, but secondly and more importantly, when it comes to creating photorealistic renders in ProDESKTOP, there are only two ways of settings material and colour properties; the first being face by face, and the second being component by component. With this in mind, it makes a lot more sense to make a new component for every different material encountered in a part, because it speeds up the processing of setting colour and material properties by orders of magnitude - selecting and setting properties for individual faces, particularly on complex components (as many of mine are) would take huge amounts of time, whereas a component can be selected once, its properties set, and these properties are applied to all copies of the component in the model.

Overleaf I have included a picture of the majority of the components created. However, this time when I say component I actually mean things such as an entire resistor assembly, comprising of ceramic body, colour bands, and pins, or an entire LED (as opposed to the anode and cathode inside the lens, the lens and the pins). The reason for doing this is that a picture of a single resistor pin or band is not very interesting. As an example of the difference, pictured below is the power button's constituent parts, and the assembly. There are infact 157 ProDESKTOP design files for this project - a few being assembles of sub components - the rest being components.

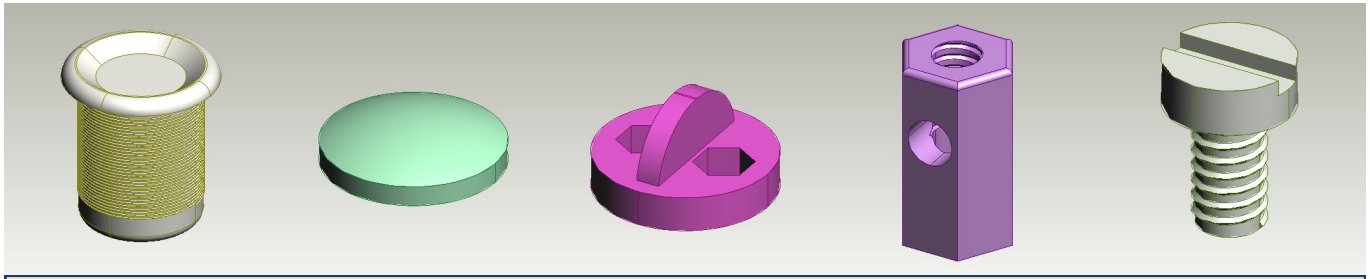


Figure 44.3 - Power button components

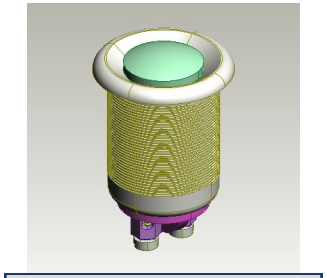
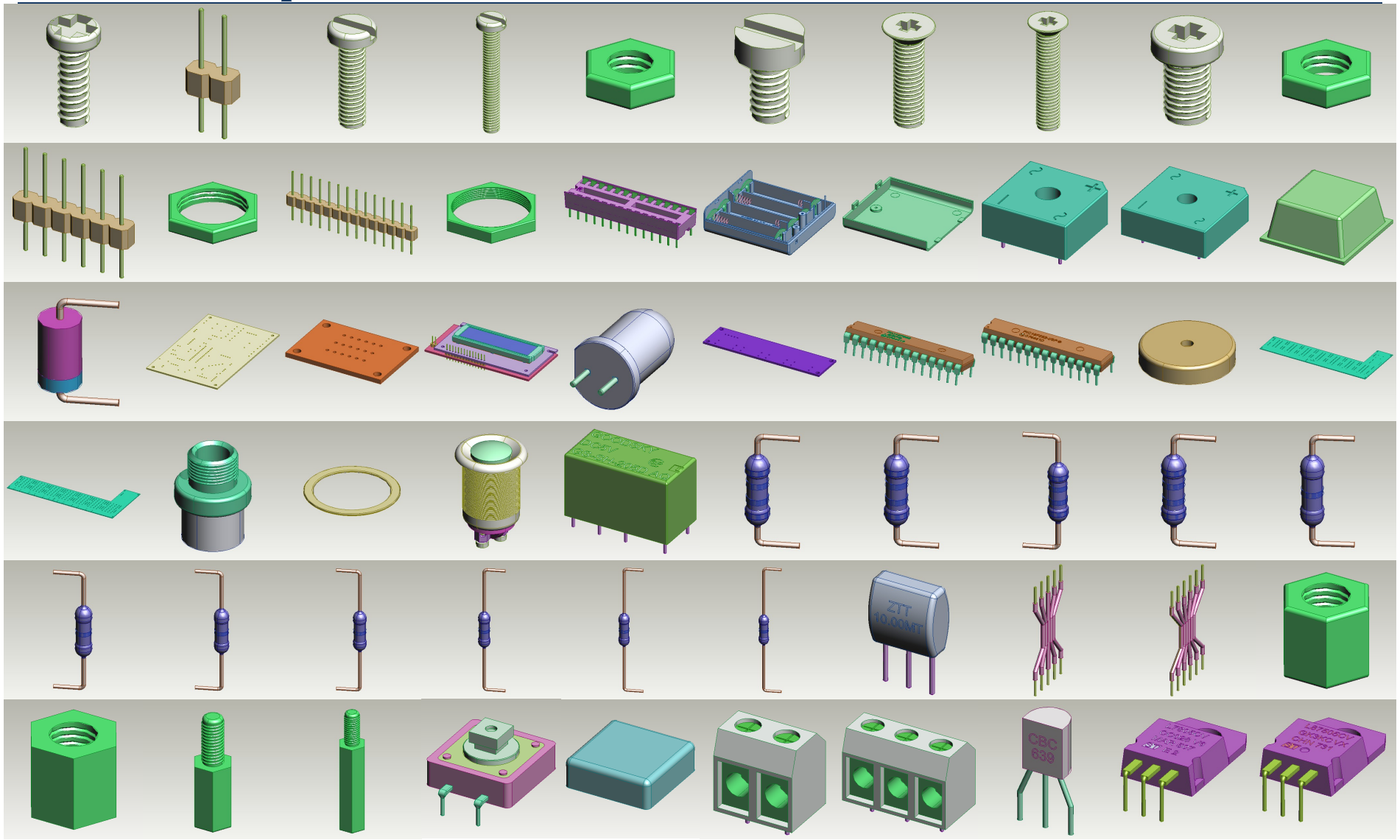


Figure 44.4 - Assembly

Case Development



Case Development

Pit Board Assembly

Prior to creating the actual case design model, I assembled a model of the pit boards and saved this as a separate file. The reasoning behind this is that ProDESKTOP processes the position of every component in a design when you update any single component. The consequence of this is that the program becomes very sluggish (think five seconds per frame here!) when dealing with designs with hundreds of components in. By creating a separate pit board assembly, I was able to avoid such a massive performance hit when assembling the case design model, because ProDESKTOP treats the pit board assembly component as a single solid, rather than processing each subcomponent. Nevertheless, I still encountered processing delays when creating the case design model when using this pit board assembly, because there are so many components in the case design model in addition to the pit boards. Figure 44.5 shows the pit board assembly.

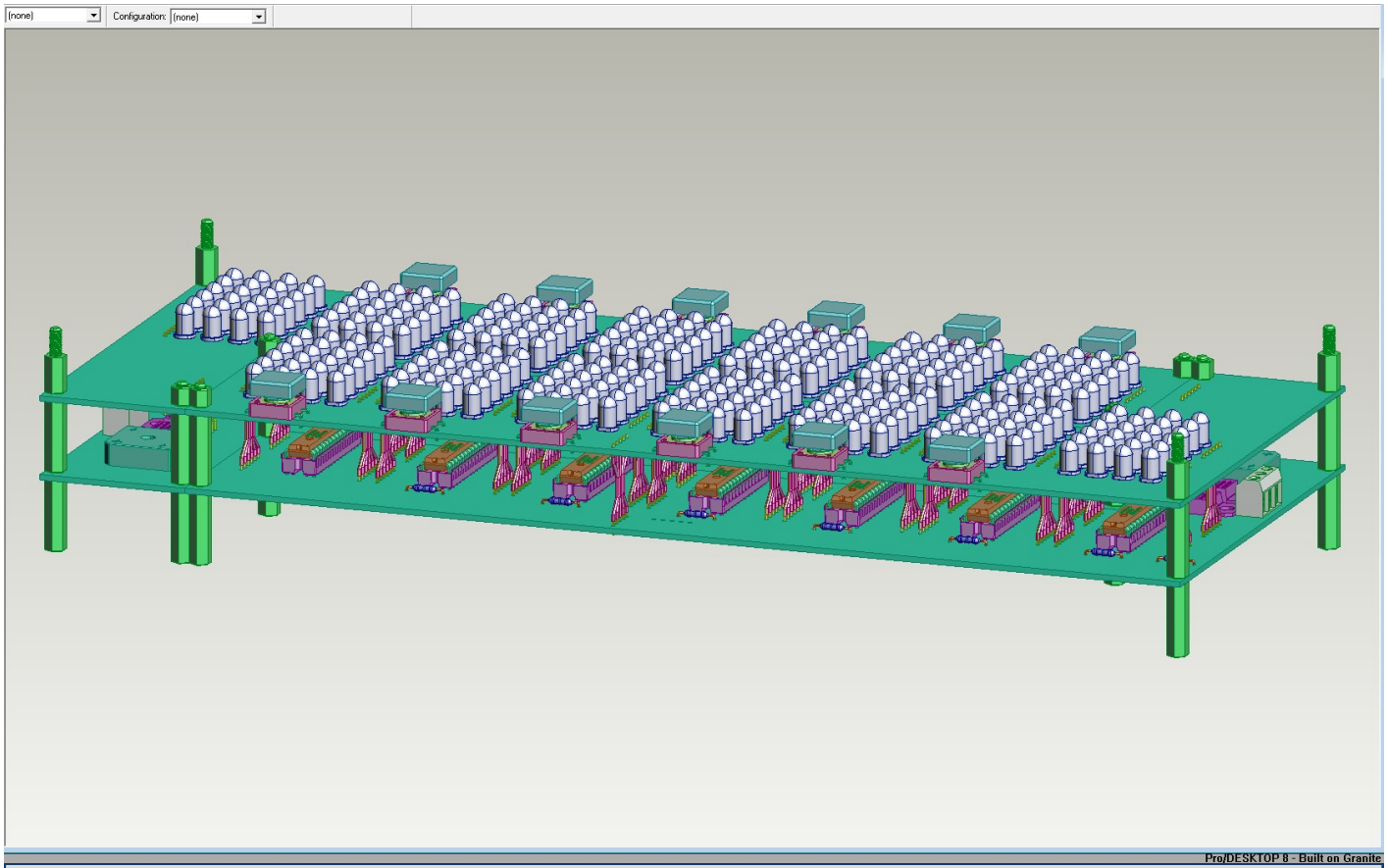


Figure 44.5 - Pit board assembly

Case Development

Case Panels

Shown in figures 44.6 and 44.7 respectively are engineering drawings of the lower and upper case panels.

All linear dimensions are either marked or can be derived from those marked. I have not marked the dimensions for mounting holes on to avoid cluttering.

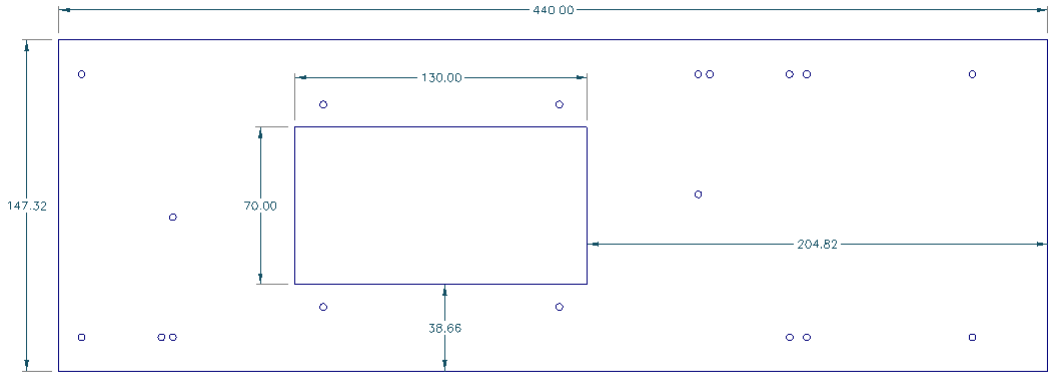


Figure 44.6 - Lower panel

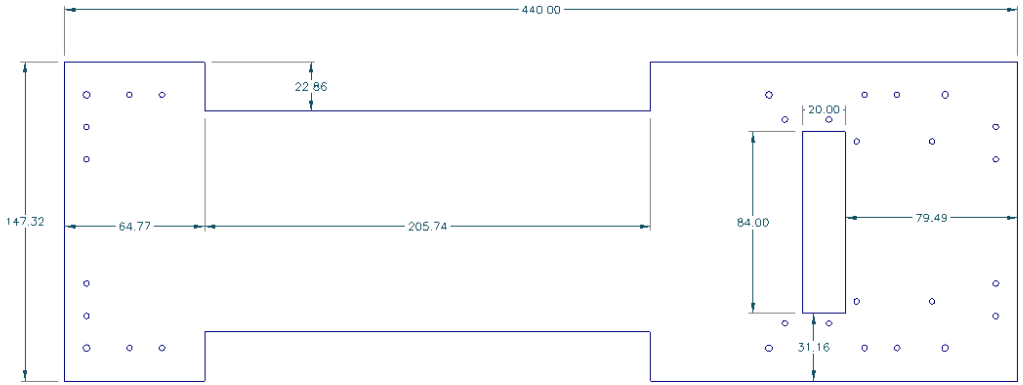


Figure 44.7 - Upper panel

Case Development

Case Panels

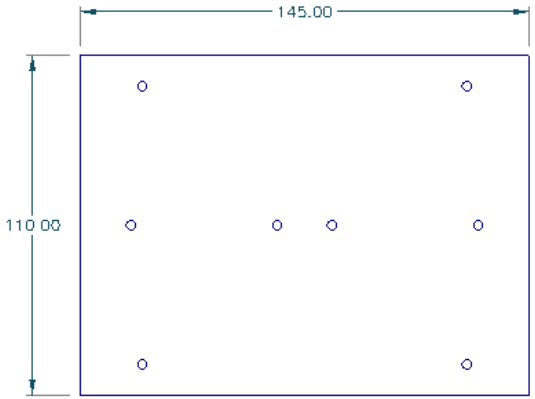


Figure 44.8 - Battery box mounting panel

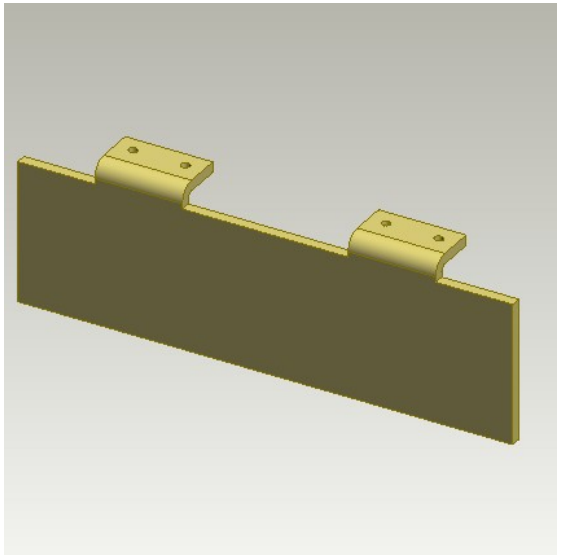


Figure 44.9 - Pit end side panel

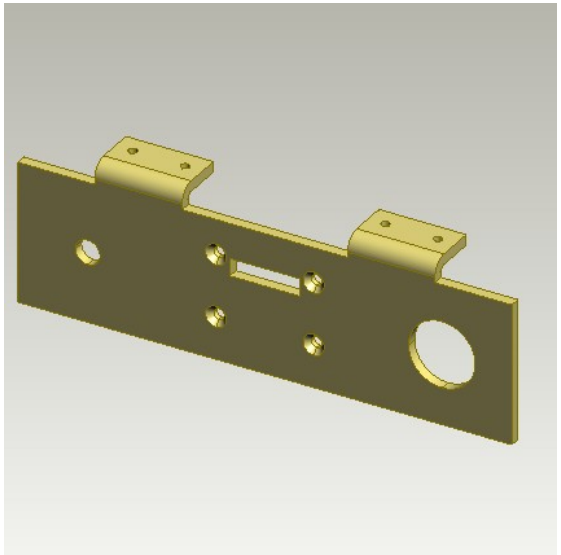


Figure 44.10 - Game logic end side panel

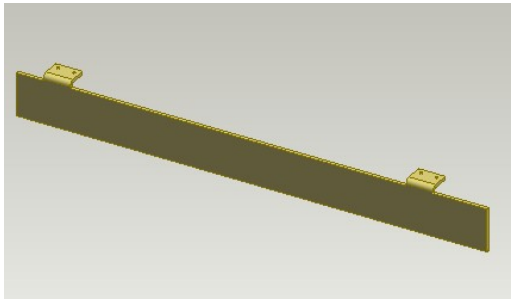


Figure 44.11 - Player one side panel

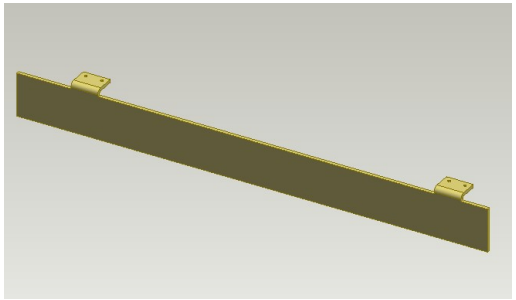


Figure 44.12 - Player two side panel

Case Development

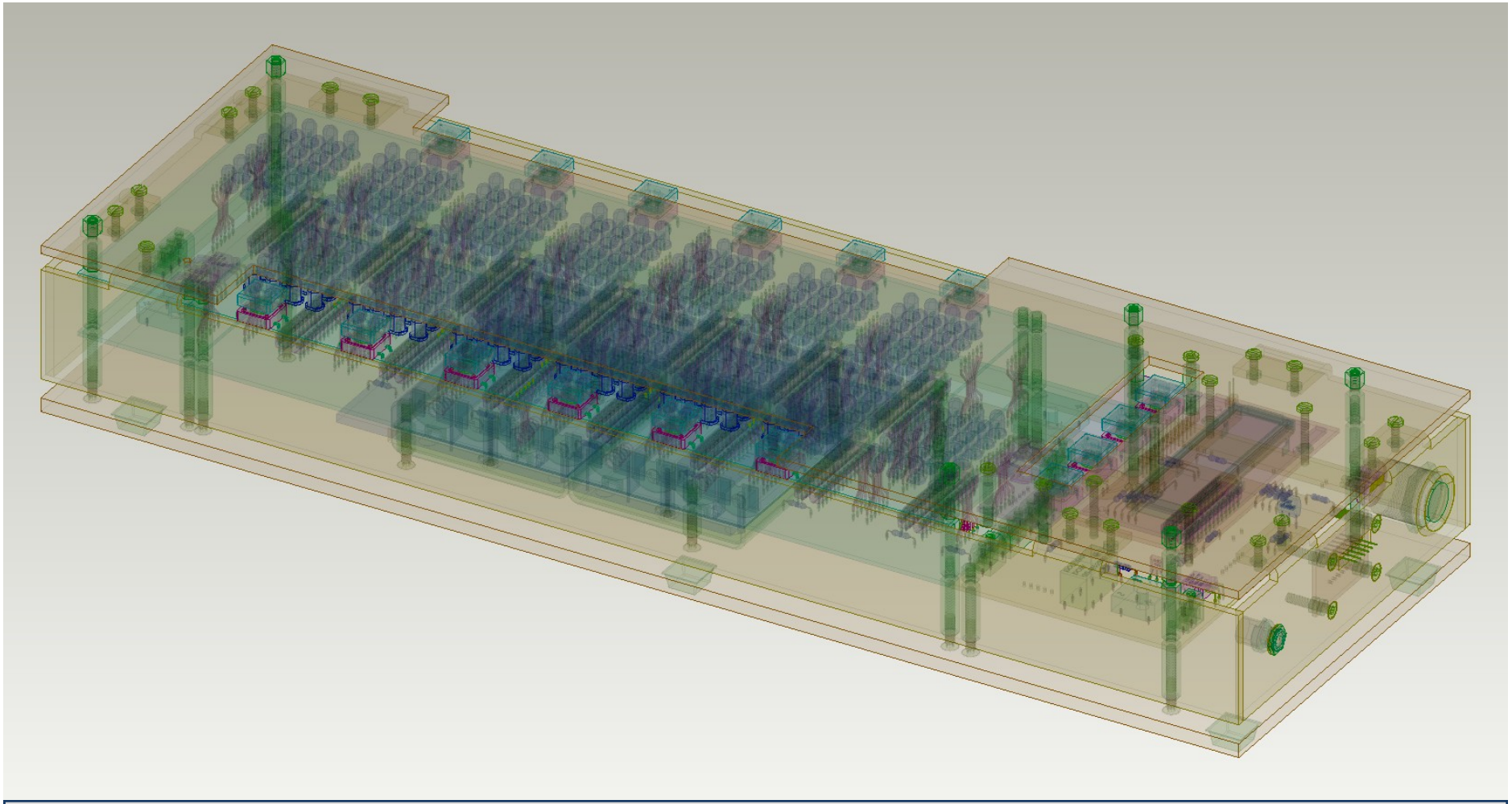


Figure 44.13 - Trimetric view - Transparent

Case Development

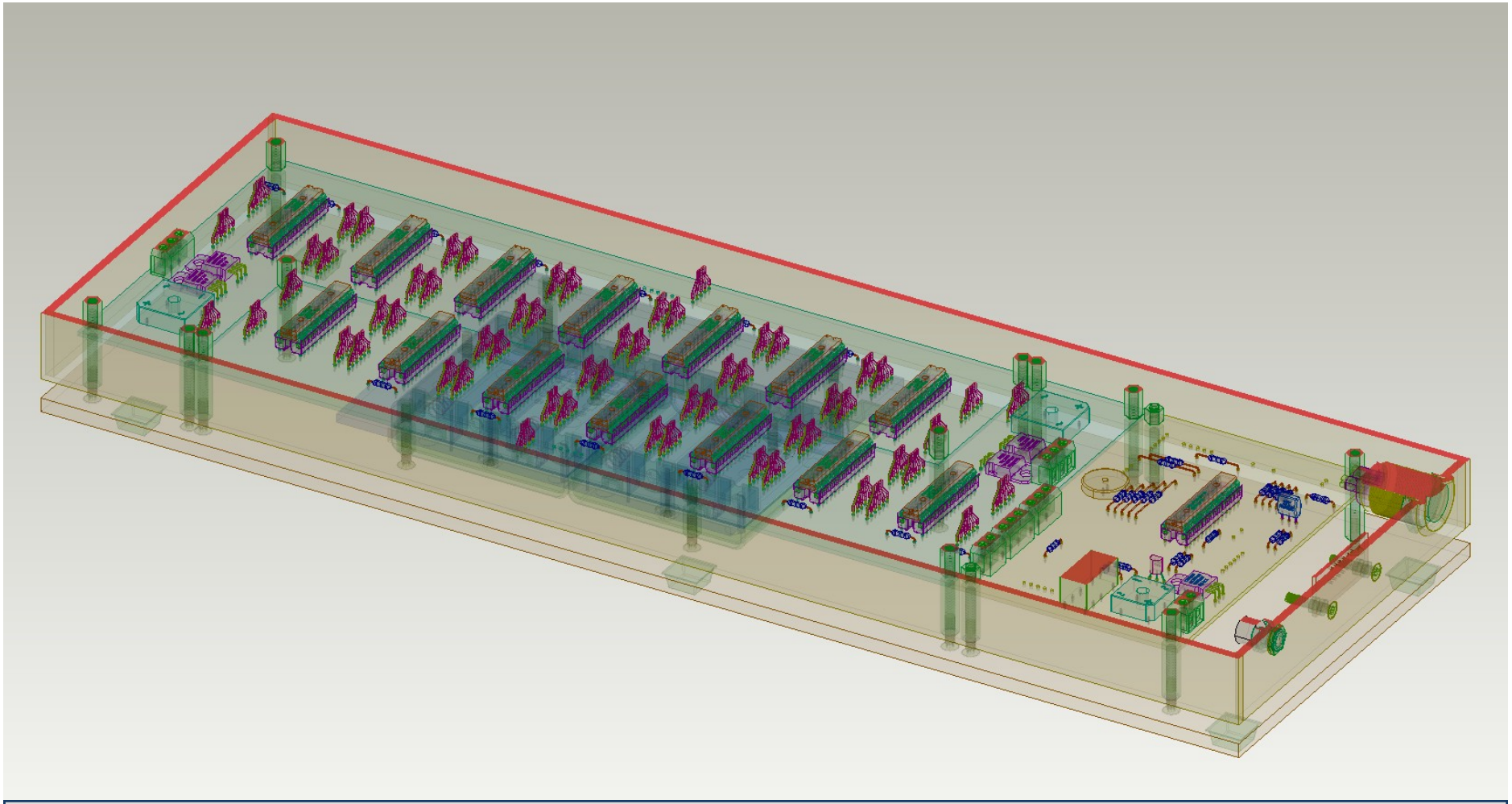


Figure 44.14 - Trimetric view - Basal cross section - Transparent

Case Development

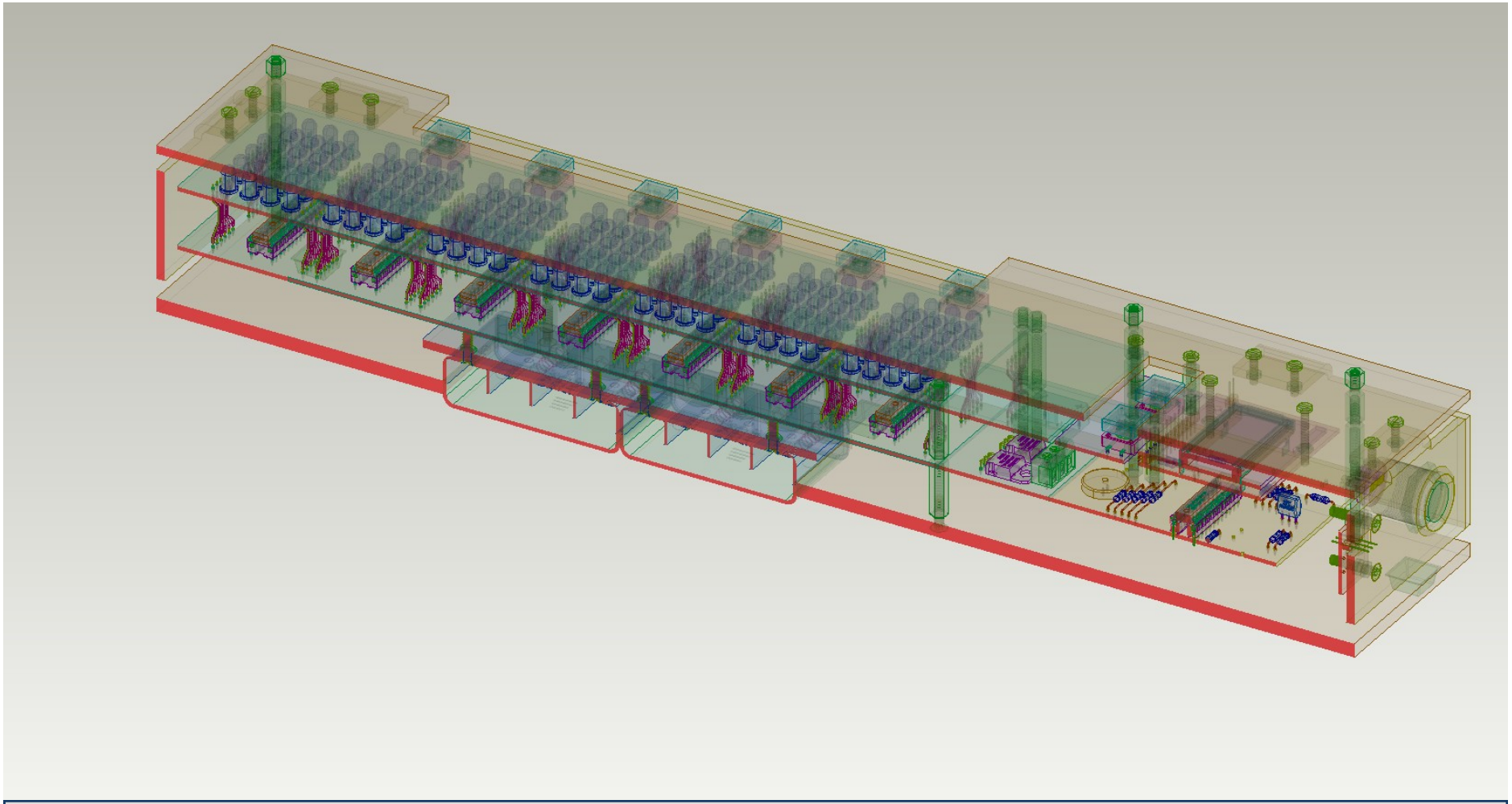


Figure 44.15 - Trimetric view - Frontal cross section - Transparent

Case Development

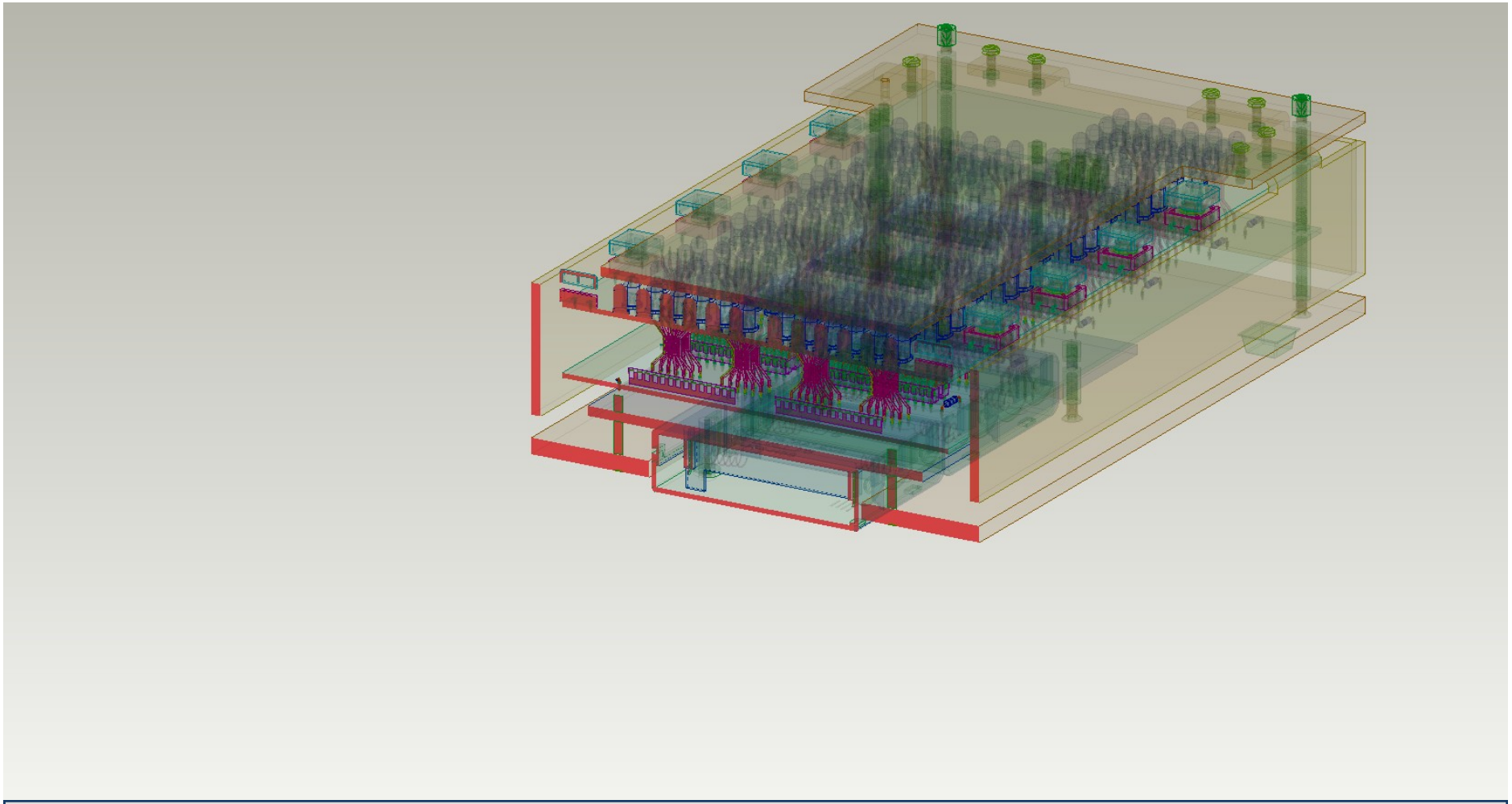


Figure 44.16 - Trimetric view - Lateral cross section - Transparent

Case Development

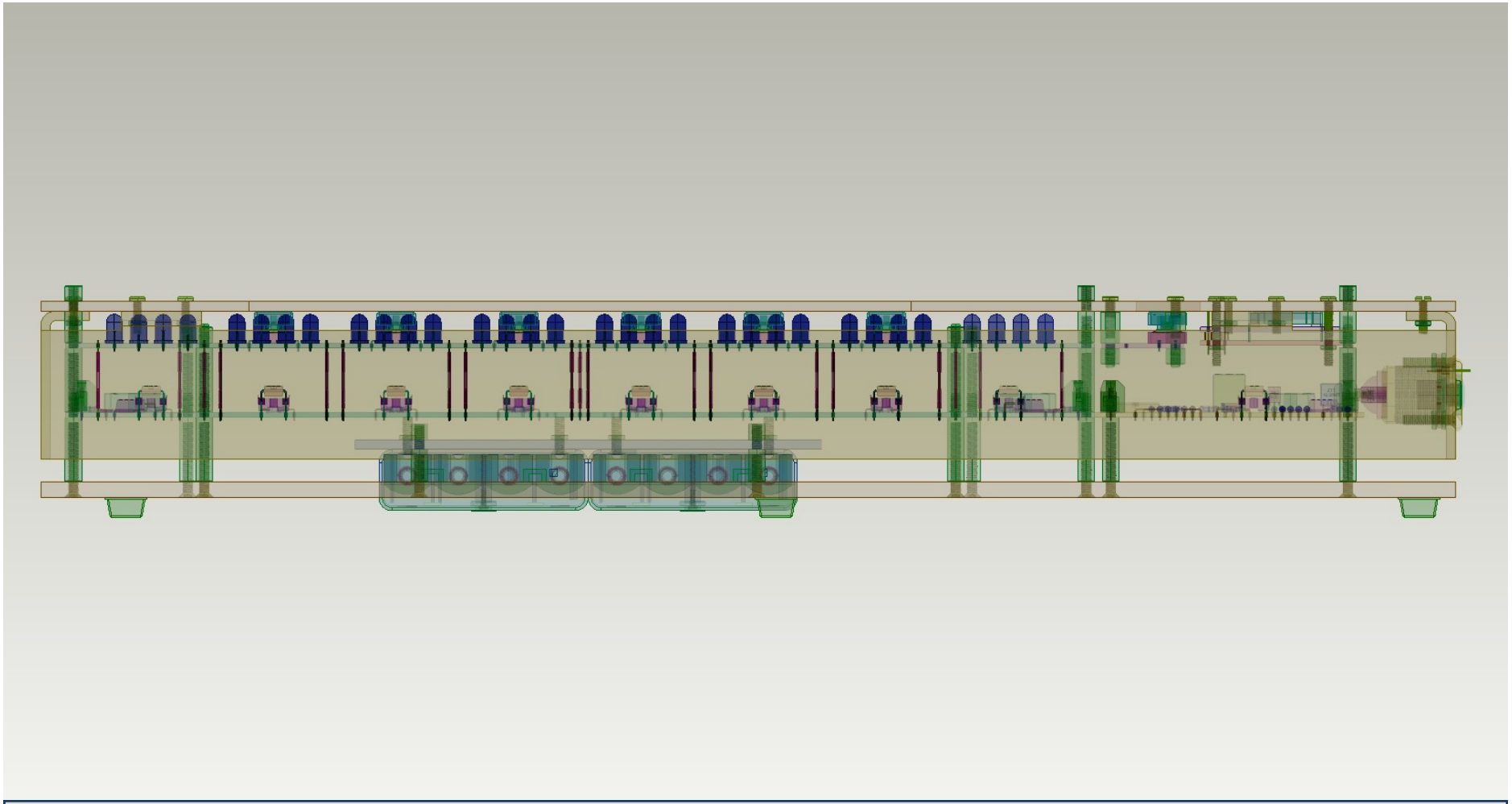


Figure 44.17 - Frontal view - Transparent

Case Development

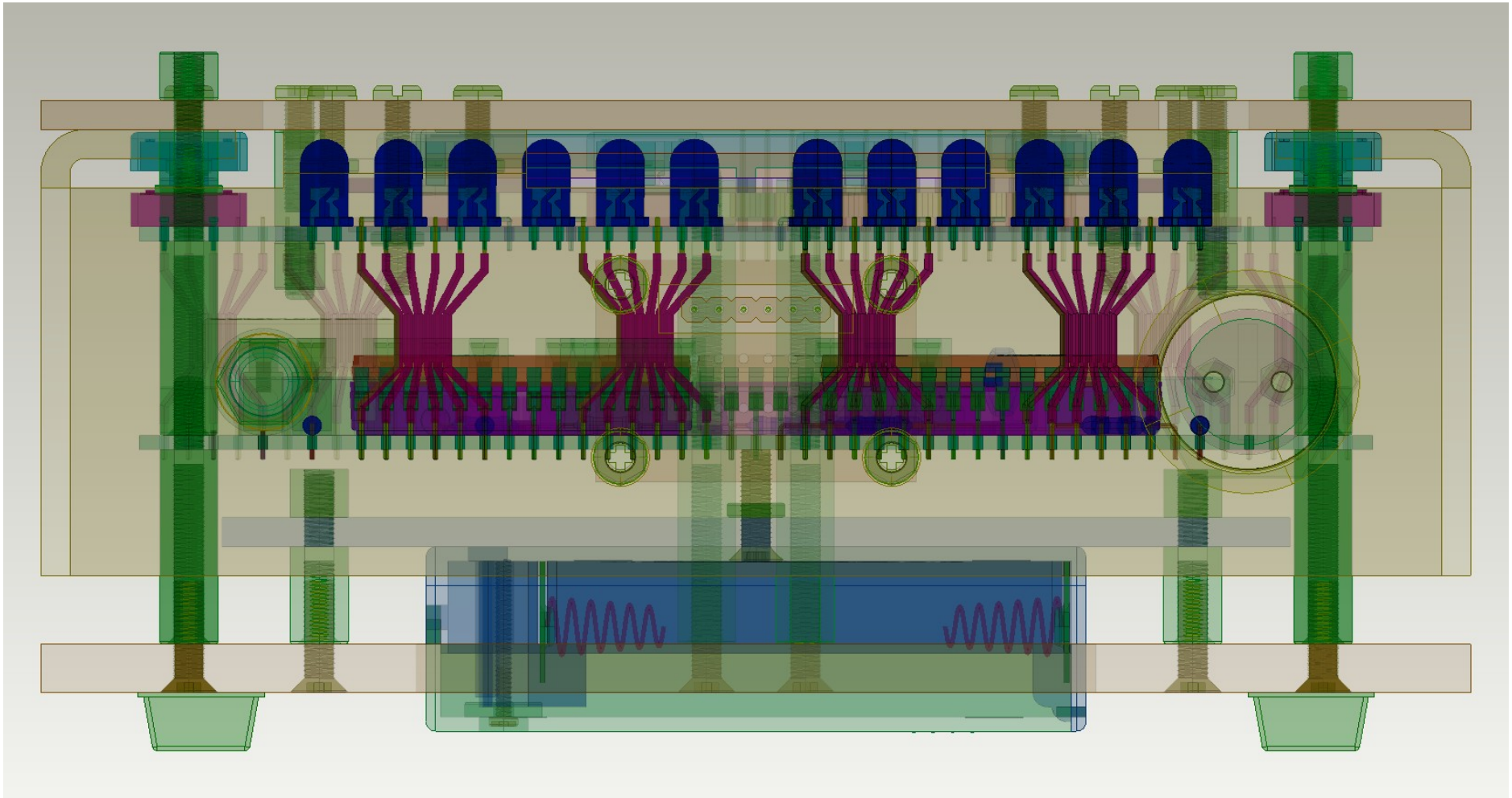


Figure 44.18 - Lateral view - Transparent

Case Development

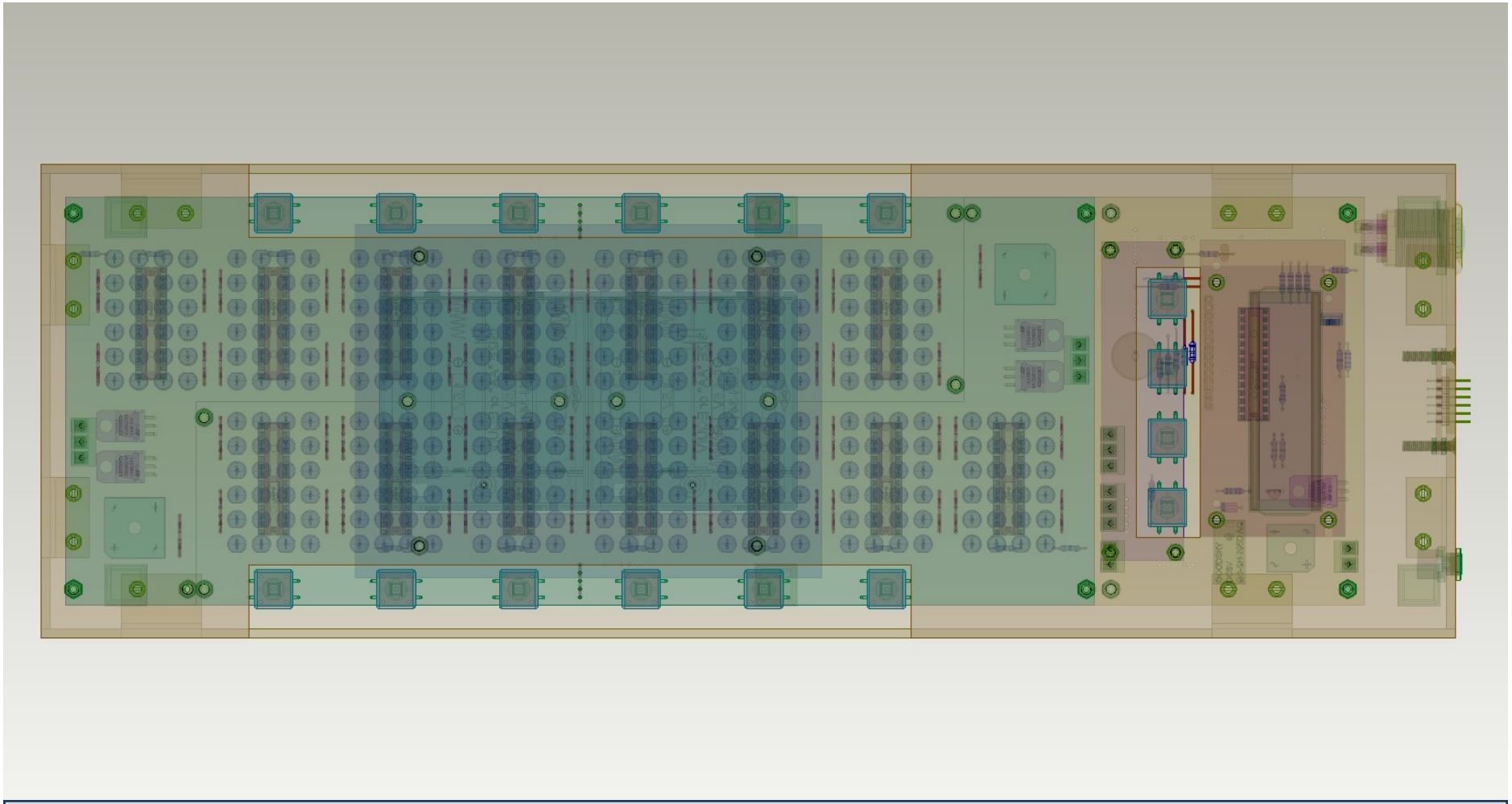


Figure 44.19 - Basal view - Transparent

Case Development

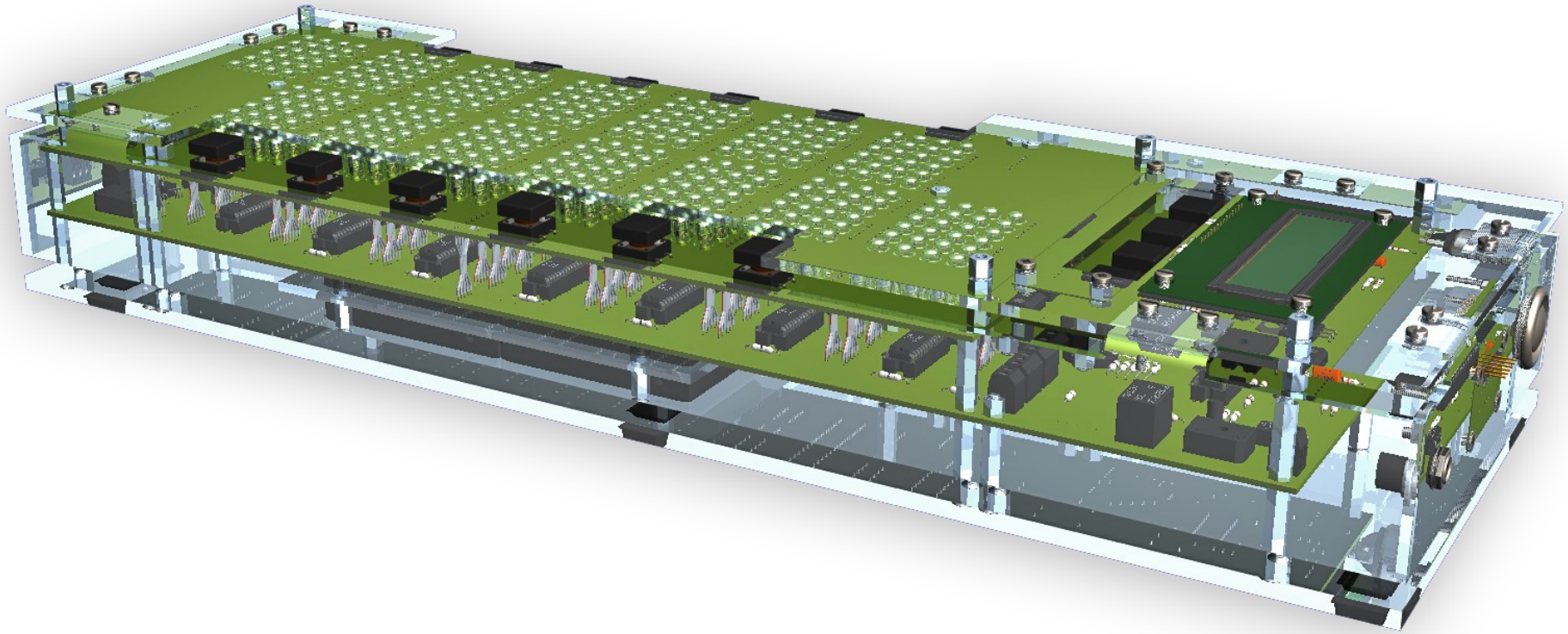


Figure 44.20 - Trimetric rendering

Case Development

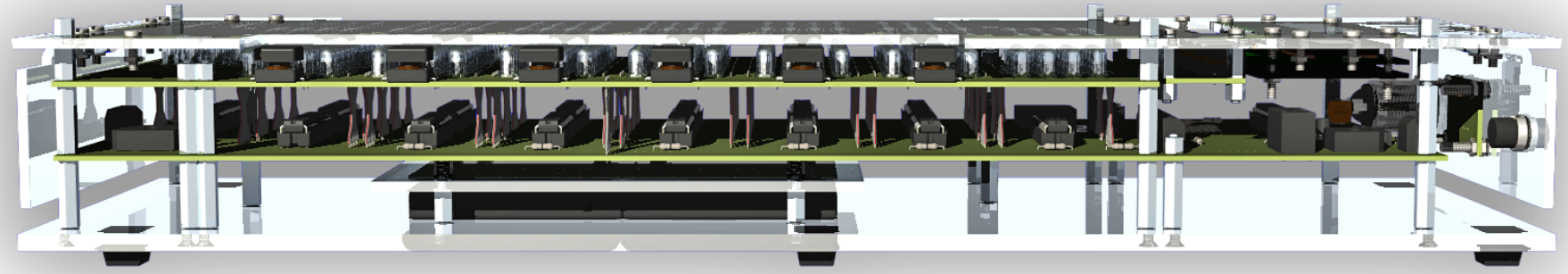


Figure 44.21 - Frontal rendering with side panels removed to minimise reflections

Case Development

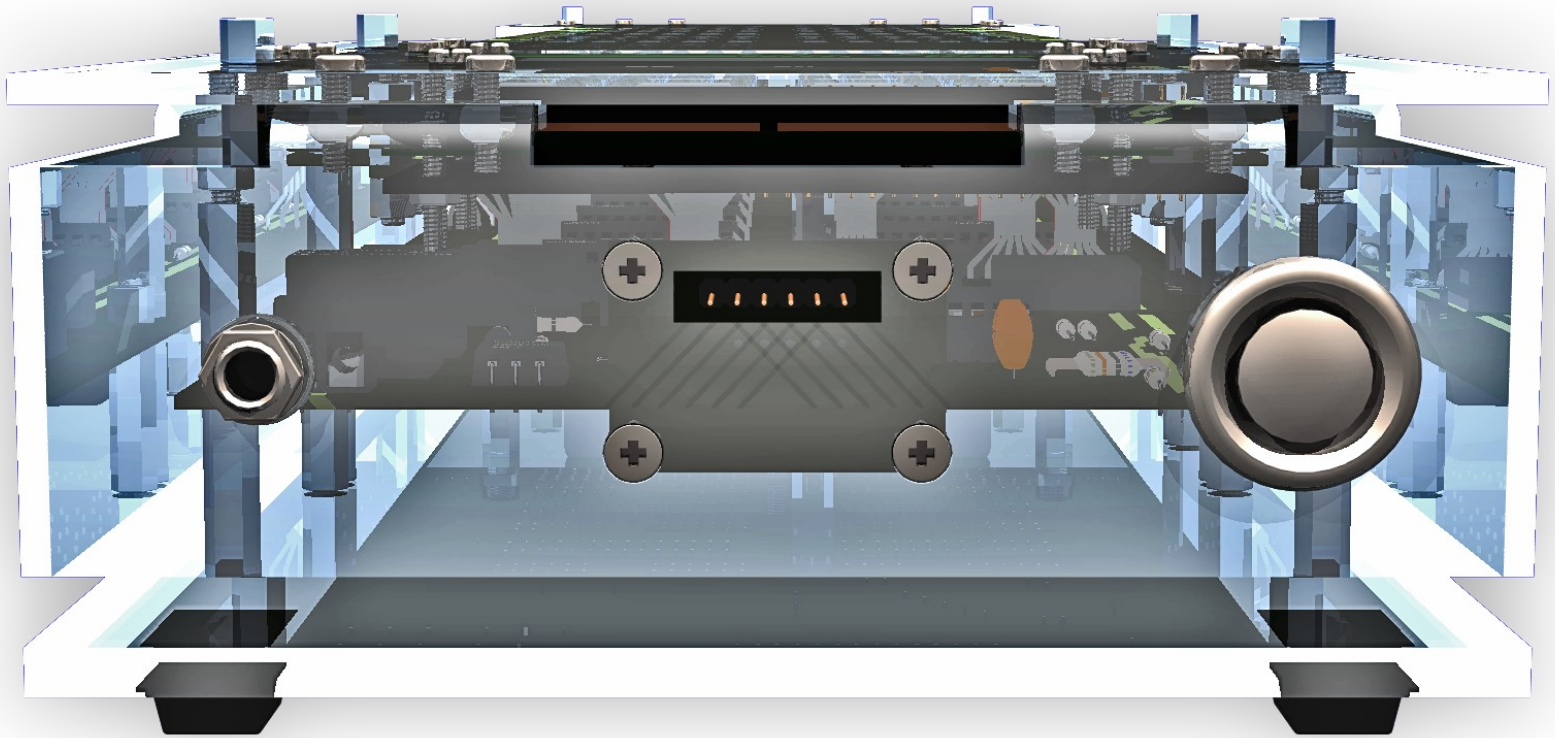


Figure 44.22 - Lateral rendering

Case Development

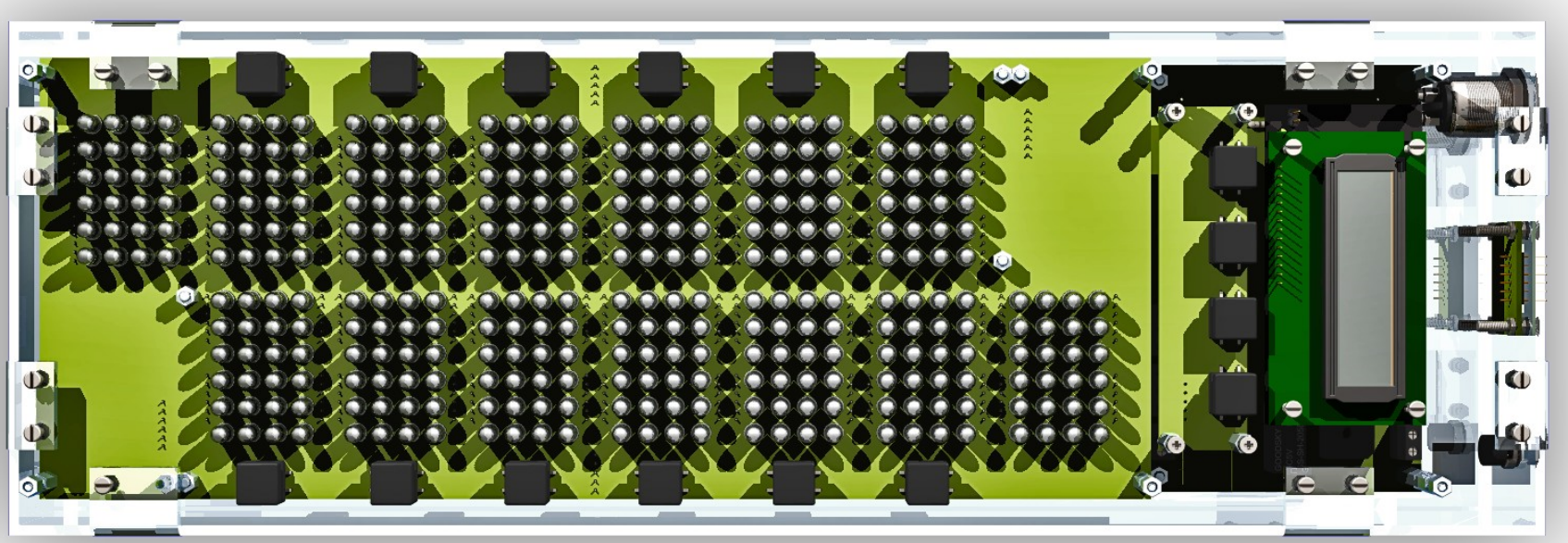


Figure 44.23 - Basal rendering with upper panel removed to minimise reflections

Case Development

Development Evaluation

As I have only designed, and not yet produced the case, I am not yet in a position to evaluate the design itself - just the process.

Short of adding solder blobs, PCB tracks, inter-board wires, and components on the back of the LCD to the model, it cannot really be any more accurate and detailed than what I have done. In this regard, the design is perfect.

Aside from the slowness of ProDESKTOP due to the model complexity, the only real problem I had was when it came to rendering; the album mode of ProDESKTOP leaves something to be desired when it comes to materials selection and lighting options. This is most evident with the acrylic panels - in the model they are actually set as glass because this is the only transparent material available. This caused all sorts of problems due to the difference in optical properties - it was a real challenge to create renders in which the components inside were reasonably visible (not hidden behind the reflection of an 'acrylic' panel) and the limited number of lighting options didn't help the issue. Another example is the PCBs - there is no fibre glass material, so after some trial and error found graphite to be the best substitute.

In the end, I concluded that the only two ways of making the components inside visible are either a) using default lighting, which is not photo-realistic or b) hiding the 'acrylic' panel blocking the view. Note however that the lateral rending is done with room lighting and the acrylic panel is not hidden - it didn't look right without the panel and I/O.

Case Production

To CAMM or not to CAMM

As mentioned at the start of case development, the design is simply too big to fit on the CAMM machine-bed and mill parts in one run without turning them around - which would lead to inaccuracies. I decided to either CAMM everything, or CAMM nothing - the reasoning behind this being that I've found the majority of time spent with CAMM is refamiliarisation, so it would make sense to get the biggest return on this investment. However, I decided not to use CAMM.

Rectangular panels

The first step was to draw up a cutting list for the seven panels - all are 3mm acrylic with the exception of the lower panel which is 5mm for rigidity. These were cut with the bandsaw.

Rectangular panels with lines on

Next I marked up the lines along which cuts were to be made or holes added. I did this with a set square, ruler and medium tip permanent marker , using the ProDESKTOP design as reference. All holes could then be marked from points of intersection - more accurate than trying to simultaneously ensure two distances are correct.

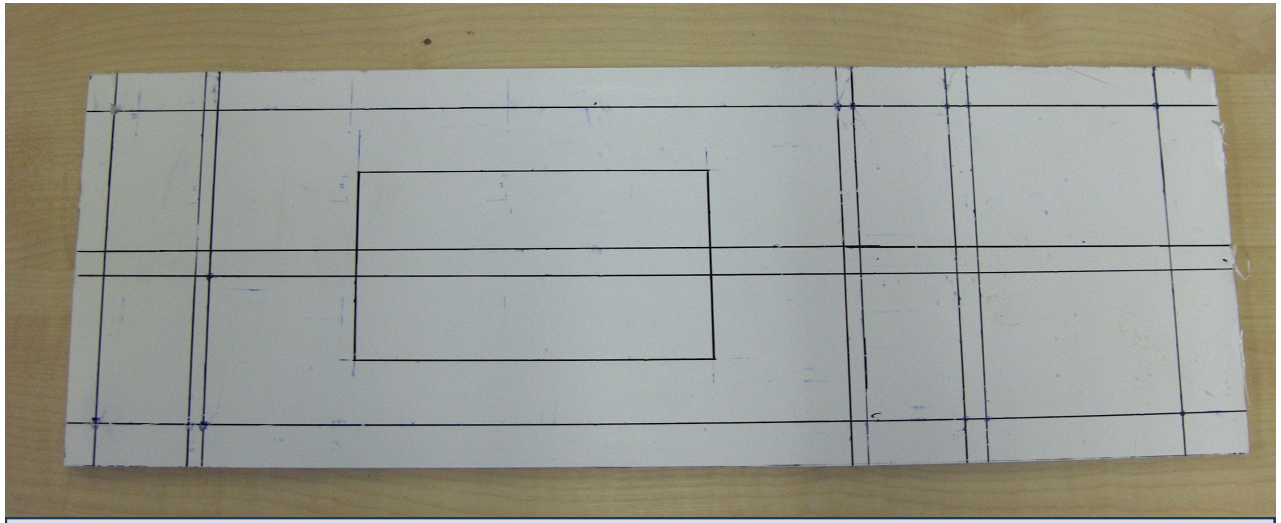


Figure 45.0 - Lower panel with mark-up on

Case Production

Rectangular panels with holes in

The holes in the lower panel are 3mm diameter for the 3mm diameter, countersunk, crosshead, 10mm long screws used to attach it to the 20mm standoffs of the pit boards and game logic board. I drilled these using a pillar drill, and then changed to a (powered) hand drill for the countersinking, because it wastes more slowly and so would reduce the chance of countersinking too deep (I don't think the travel restriction on the pillar drill would have been accurate enough for this). After having done this, I mounted the pit boards and game logic board on the lower panel to check they fitted. This is shown in figure 45.1.

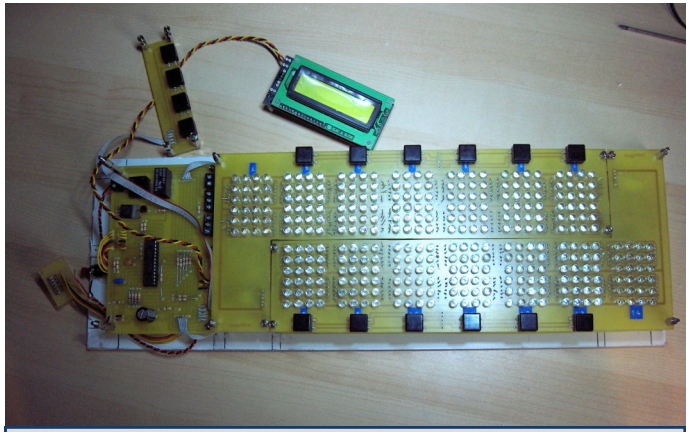


Figure 45.1 - Pit boards and game logic board mounted on lower panel

Panels with cut-outs

For the panel cut-outs (for pit buttons, battery boxes, menu buttons, and ICSP header pins), I used the hegner saw. For the internal cut-outs, I first drilled a 3mm hole that I could then put the blade through. These are shown below.

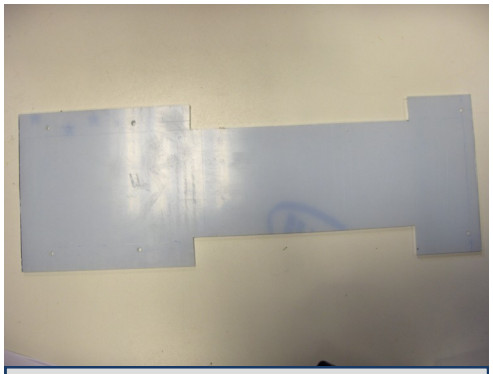


Figure 45.2 - Upper panel with pit button cut-outs

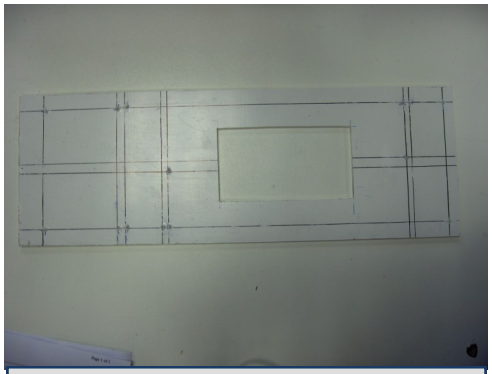


Figure 45.3 - Lower panel with battery box cut-out

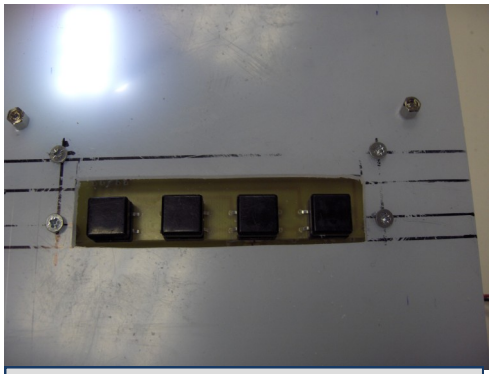


Figure 45.4 - Upper panel with menu button cut-out

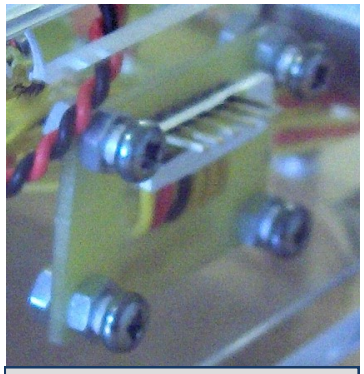


Figure 45.5 - I/O panel with ICSP header cut-out

Case Production

Edge smoothing & polishing

To tidy up and smooth the edges of the panels, I used rectangular files (primary and then secondary cut) for the outer edges and those of large enough internal cut-outs. I used needle files for the ICSP header cut-out as it was too small for rectangular files. Having done this, I used fine wet and dry paper with a sanding block to prepare the edges for polishing with the plastic polishing wheel.

Side panels

I created the tabs such that they would be 3cm long to start with, and I could then shorten them to fit the case. The body of the side panels is 4cm wide, so I started with 7cm wide panels and cut out from the upper corners and centre with the hegner saw so that I was left with the required tabs. I then filed and polished them in the same way as the upper, lower and battery box panels, before using the strip bender to heat them at the bending point until soft enough to bend using an MDF block and the edge of a work bench as a former to create the required radius.

Having done this, I compared the tabs with the upper panel in situ on the pit boards to see how much acrylic would need trimming. This I marked on with a permanent marker and then cut away using the hegner saw - which was tricky due to the angle. I then refiled these edges before again holding them to the upper panel to mark on the mounting holes, which I drilled with a (powered) hand drill as the angle meant it couldn't be done with a pillar drill.

For the end panel with I/O (power socket, ICSP header and power button), I drilled the power socket hole and ICSP header mounting holes with standard drill bits. However, the power button required a 19mm hole which was too big to drill using a standard drill bit as the stress would have shattered the panel. Instead, I used an 18mm spade bit and then used a reamer to increase it to 19mm so the power button would fit.

Case Production

Case assembly

With the panels complete, it was time to assemble the case. As the construction is centred around the PCBs, this was more a job of building it around them as it was a job of installing them in it. The following table details the screw type used for each fixing:

Fixing (between)	*Diameter (mm)*	*Length (mm)*	*Head profile*	*Head pattern*
Lower panel, pit & game logic board standoffs, battery box panel standoffs	3	10	CS	Cross
Side panels & upper panel	2.5	10	Pan	Flat
ICSP board and I/O panel	3	15	CS	Cross
LCD & upper panel	2.5	20	Pan	Flat
Menu button standoffs & upper panel	3	5	Pan	Cross
Battery box panel & battery boxes	3	10	CS	Cross

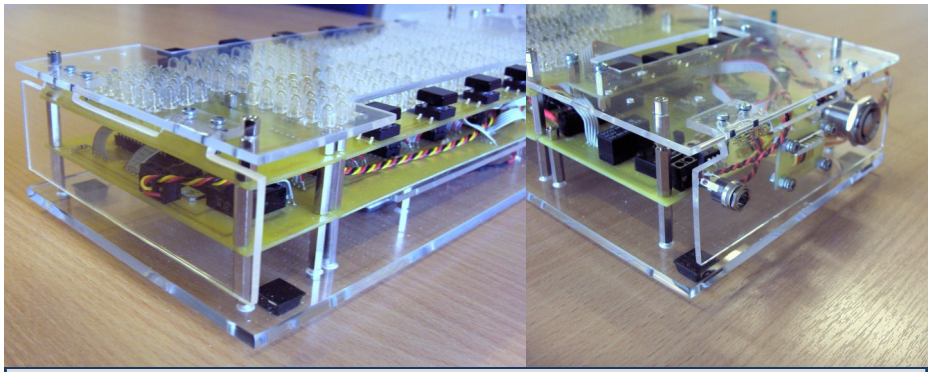


Figure 45.6 - End panels attached

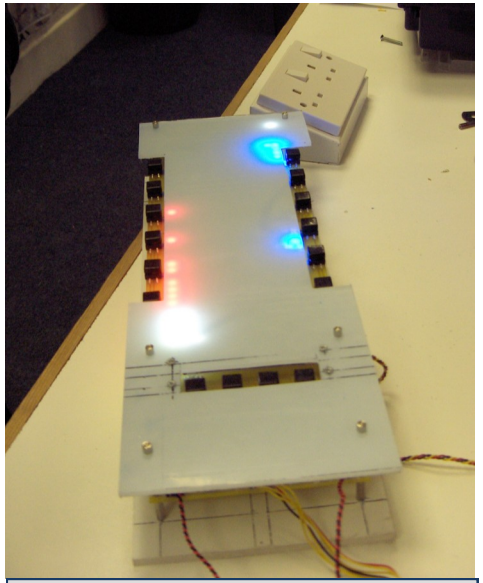


Figure 45.7 - Upper and lower panels attached

Case Production

Power supply

I connected the two battery boxes in series by soldering the positive wire of one to negative of the other, with heatshrink over the joint to prevent short circuits. I then extended the remaining two wires in the same fashion as they were not long enough to reach the terminal block on the game logic board, and using a drill twisted the pair together for neatness. The power supply system is discussed in more detail in the evaluation.

Switch remapping

I realised when developing the idea in ProDESKTOP that the game logic board would me much better rotated 180° from how I had planned for it when I created the artwork, so I designed and assembled the case like with it like that. However, as a result the two ribbon cables that go to the pit board switches were cross over and making the case look messy, so I desoldered them and swapped them over. The case looks much tidier as a result. However this did mean that I had to update the column and row mappings that I had created for the switch arrays. This is detailed in the programming section that follows case production.

General tidy up

I shortened the length of the LCD and ICSP wires such that they matched the length of the power socket and switch wires; the amount by which the upper portion of the case can be separated from the lower is limited by the minimum wire length, so the excess was doing no good. I also wrapped all wires in spirowrap, except ribbon cables.

Case Pictures

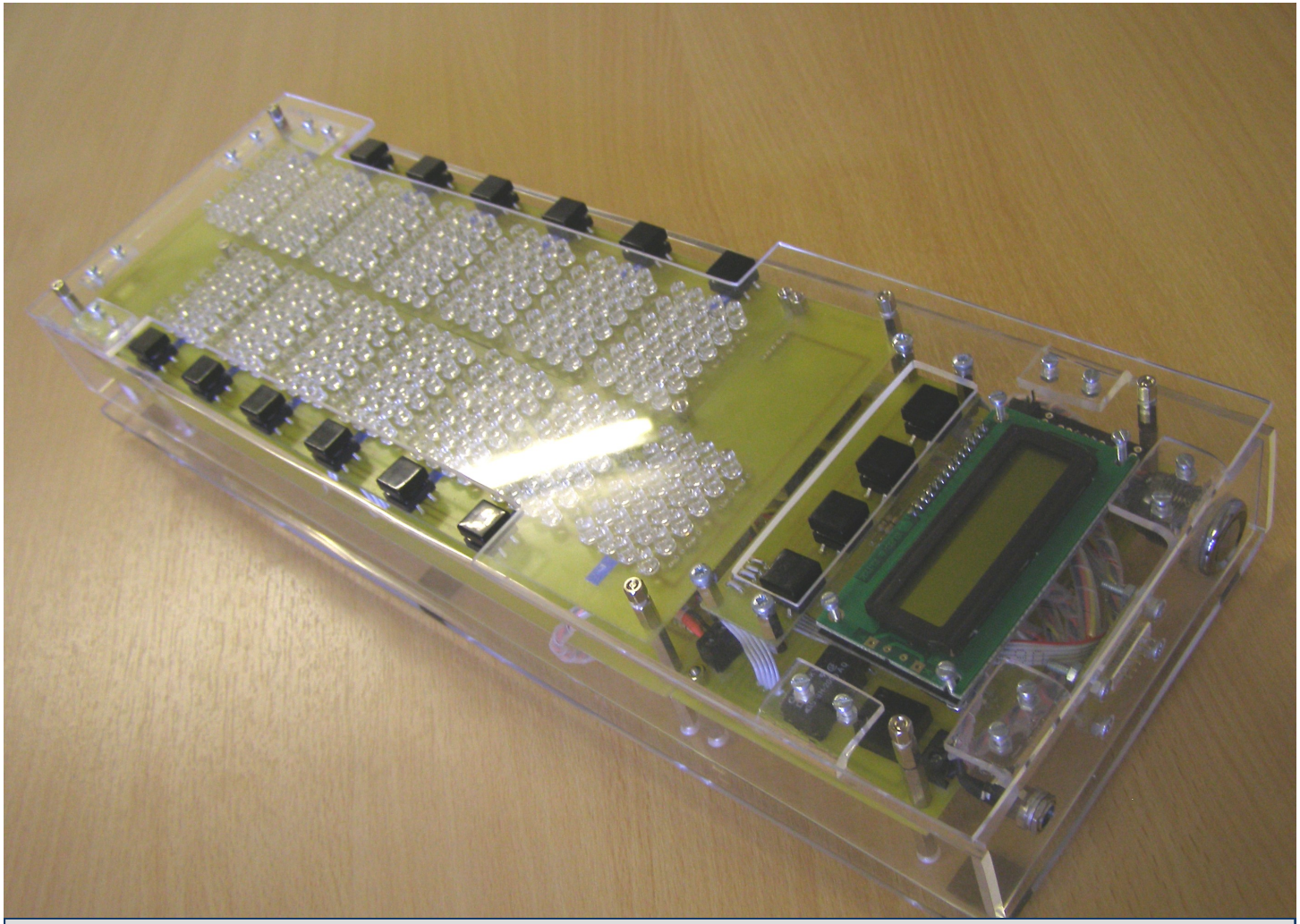


Figure 46.0 - Trimetric photograph

Case Pictures

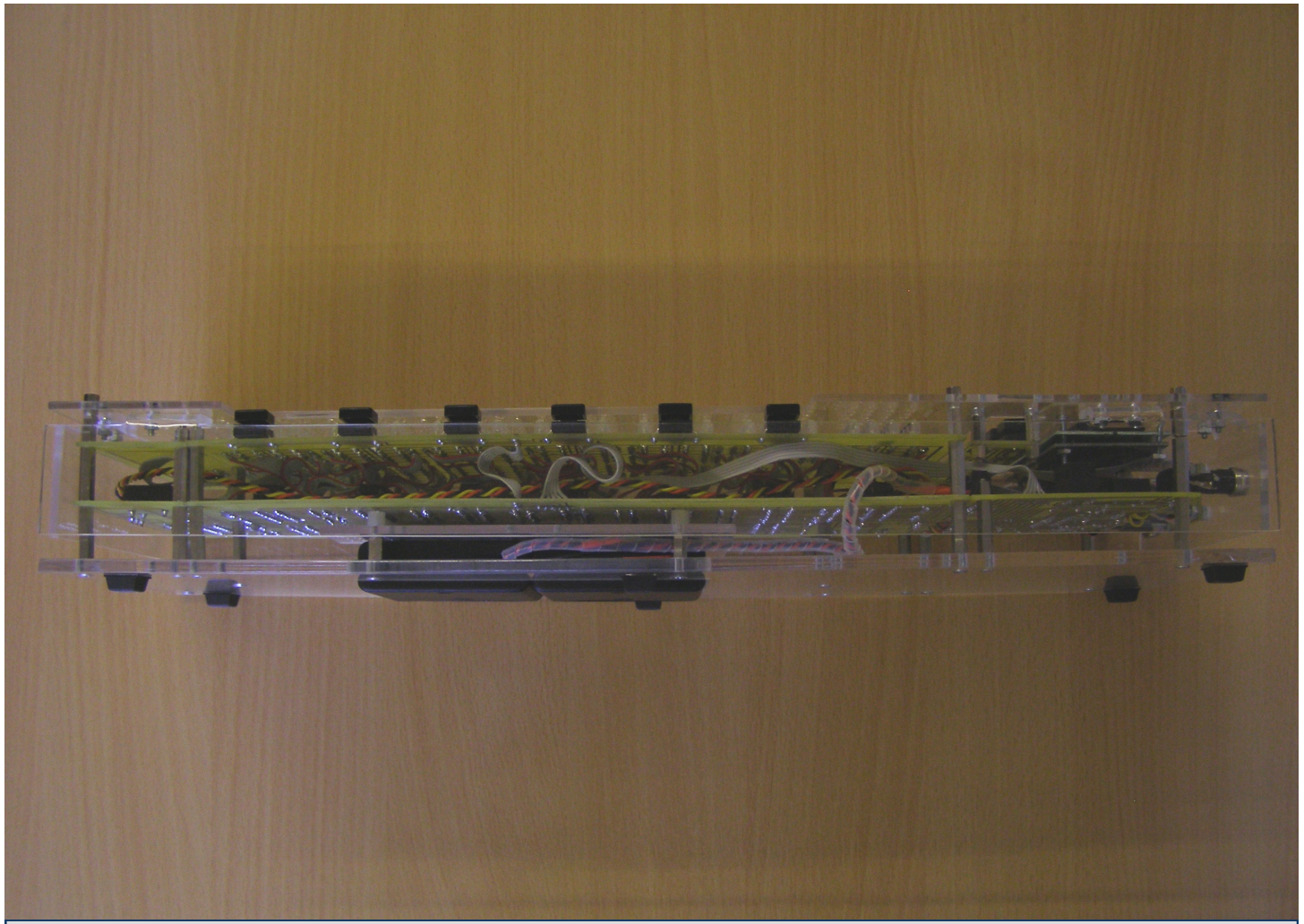


Figure 46.1 - Frontal photograph

Case Pictures

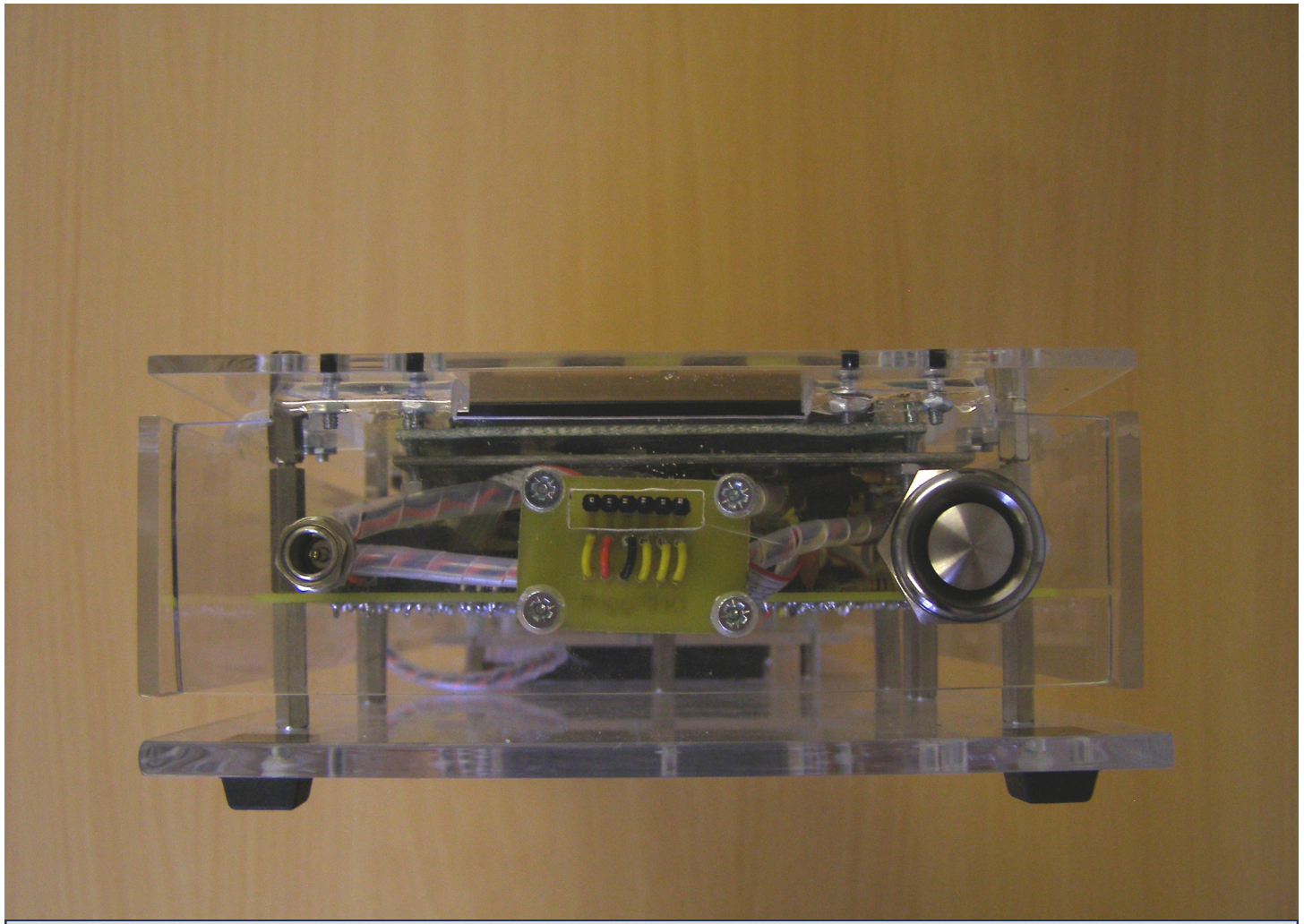


Figure 46.2 - Lateral photograph

Case Pictures

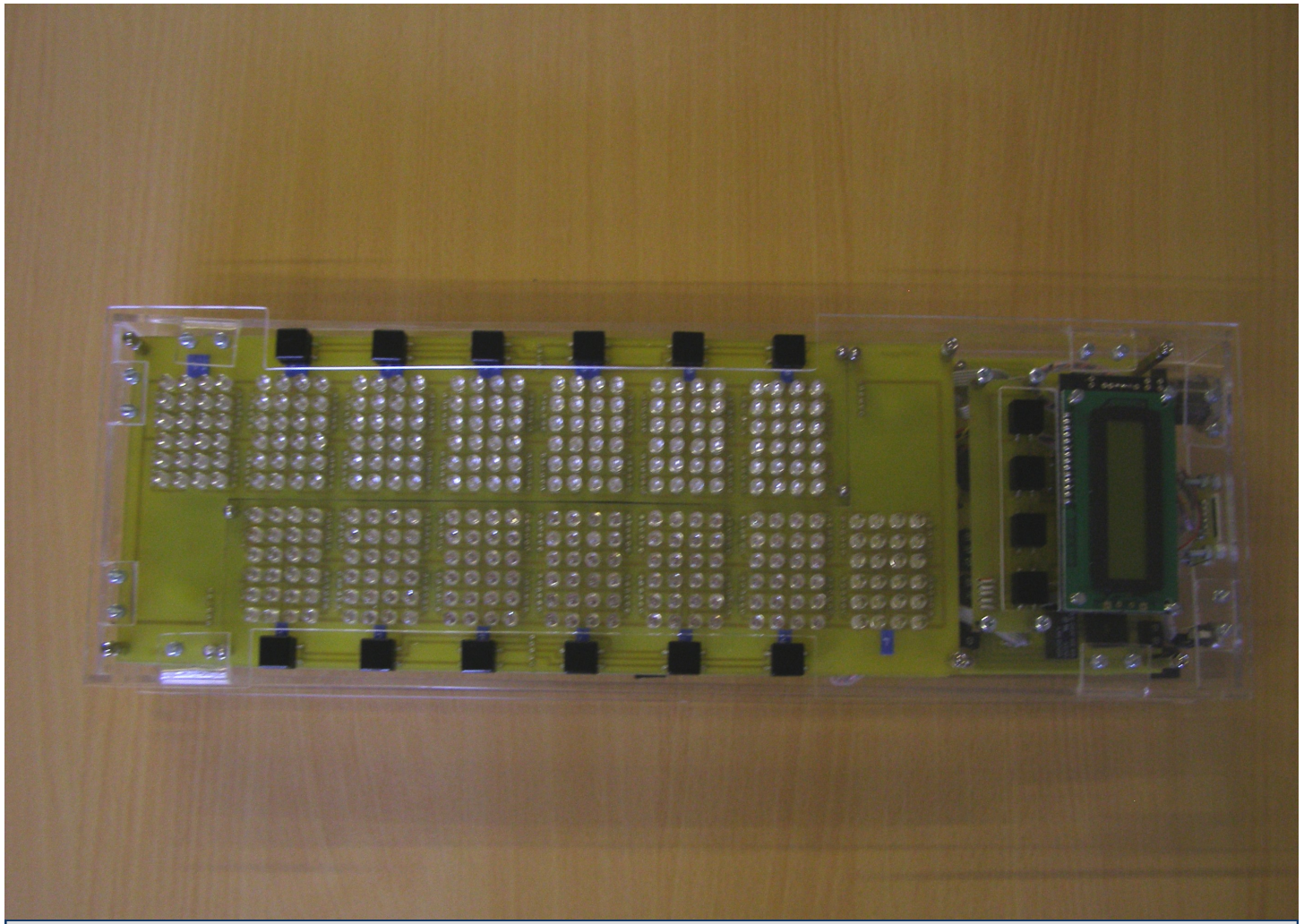


Figure 46.0 - Basal photograph

Time Usage & Unfortunate Circumstances

ID	Task Name	Start	Finish	Duration
1	Christmas Holidays	15/12/2007	06/01/2008	23d
2	Spring Half Term	09/02/2008	17/02/2008	9d
3	Easter Holidays	20/03/2008	06/04/2008	18d
4	Project setup & initialisation	01/11/2007	01/12/2007	31d
5	Research	01/11/2007	10/04/2008	162d
6	Register control protocol & code development	03/12/2007	15/01/2008	44d
7	Schematic & Artwork development	16/01/2008	22/02/2008	38d
8	PCB production	04/02/2008	22/02/2008	19d
9	Game logic code proof of concept	20/02/2008	25/02/2008	6d
10	Case dimensioning & model creation	26/02/2008	07/03/2008	11d
11	Case production	10/03/2008	19/03/2008	10d
12	Time allocated for addition of LCD menus & AI player to game logic code	20/03/2008	06/04/2008	18d
13	Completion of documentation	07/04/2008	10/04/2008	4d
14	Replacement PIC18F252 ordered (x2) & completion of game logic code	07/04/2008	10/04/2008	4d

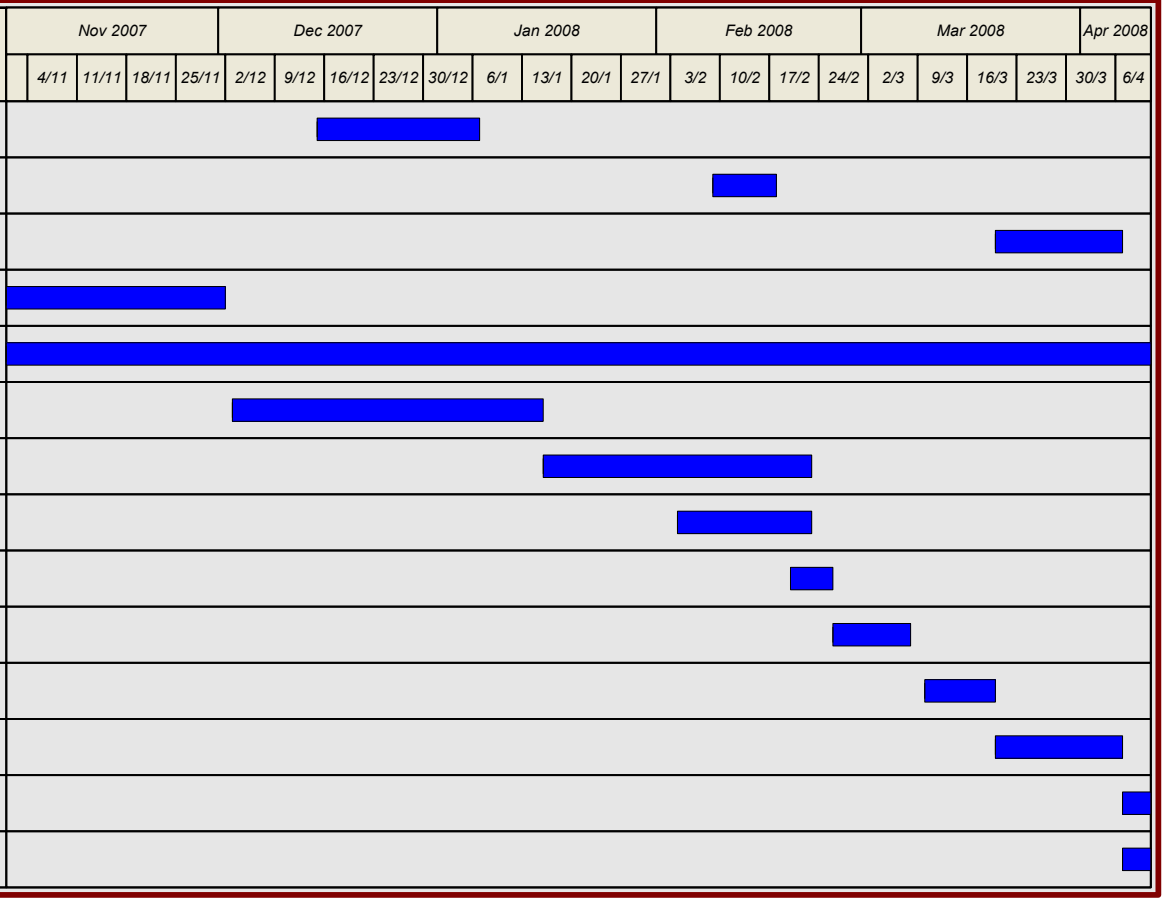


Figure 47.0 - Time usage

Time Usage & Unfortunate Circumstances

Time invested in the project

Rather than estimating the amount of time spent on the various stages of the project and summing these, I have estimated my time usage by estimating the amount of hours per week spent on the project and multiplying by the number of weeks. Although there will of course have been some variance, I have been reasonably consistent throughout with my work rate. Figure 48.0 shows the estimate.

Description	*Duration*	*Quantity*	*Total duration*
D&T lessons	35 minutes	8	280 minutes
Free periods	35 minutes	10	350 minutes
Evenings	45 minutes	7	315 minutes
Sub total:		One week	945 minutes
Total:		Sixteen weeks	15120 minutes/252 hours

Figure 48.0 - Estimate of time invested in project

The estimate of 45 minutes per evening is of course the part which introduces most inaccuracy - some evenings I would do no work on it, while one or two weeks (for example during pit board production) I spent at least four hours every night populating the boards (the lengthiest task by far was stripping and tinning the ends of 672 strands of ribbon cable for the links between register PICs and LEDs in preparation for joining the upper and lower boards - which by itself took surprisingly little time, only an hour or two).

Although I did work on the project during the Christmas, half-term and then Easter holidays, I think my weekly average would have been lower than that during term time. If my estimate is inaccurate, so that it is at least an underestimate, I am completely discounting holidays from it. From figure 46.0 it can be seen that there were 16 weeks at school spent on the project, so multiplying by my estimate of 945 minutes per week, this leads to a total estimate of 15120 minutes, or 252 hours spent on the project.

Estimate of total time invested in project = 252 hours

Time Usage & Unfortunate Circumstances

Unfortunate Circumstances

Figure 47.0 shows my actual time usage throughout the whole project. It can be compared with figures 2.0, 8.0 and 42.0 respectively to see how my plans for the initialisation, development and production stages differed from my actual time use.

My management of time has been constantly changing throughout the project, as would be expected. During project initialisation, I had very aggressively scheduled the initialisation and development stages to allow as much time as possible for the production of PCBs and casing. However, as detailed on the page title “Development Dependencies”, it became apparent to me during development that I could not develop programs and artworks and then produce PCBs and a casing. Due to the nature of the project, the development and production phases were very strongly interlinked and so development and production effectively merged into one stage.

During the development stages, the deadline for the coursework was altered and moved forward to 10th April. This is the reason that on the initialisation and development timeplans, the last task, “Completion of documentation” is scheduled for completion on 19th March - the original deadline.

The main purpose of this page is to highlight task 12 of figure 47.0 - “Time allocated for addition of LCD menus & AI player to game logic code”. I had specifically scheduled the final stages of the project such that I would take advantage of the Easter holidays that were between the original and revised deadlines - my intention was to complete case production before the holidays, making use of the workshops while available, and then complete the game logic code development at home during the holidays, as the only tool required for this was the PICKit 2 programmer.

Trouble struck when the PIC18F252 that was in the game logic board stopped communicating with the PICKit 2. I tried every thing I could think of to get it working; programming from the PICKit 2 standalone application and from MPLAB IDE, programming with it self-powered and with the 12V supply attached. I tried every available software version for both MPLAB IDE and the PICKit 2 standalone application, but nothing worked. A problem similar to that which I was experiencing was described in the release notes for MPLAB 8.02, but the suggested work arounds didn't work, so I concluded my problems were due to a different cause.

Time Usage & Unfortunate Circumstances

To verify that there was no fault with the PICKit 2 programmer, I put a spare PIC16F882 in my register prototype board and programmed it with a copy of the register code; this worked fine, confirming my suspicions that the problem was with the PIC18F252. I swapped it out for my only spare, which solved the problem; I was able to download programs once again. However this didn't last long as shortly after (a few days), I experienced exactly the same problem with the spare PIC18F252 I had put in, only this time I didn't have another spare to swap it out for.

This in itself was unfortunate, but the situation is made more severe by the fact that his happened during the Easter holidays while I was away from school and so unable to order replacements. I should clarify the situation here by pointing out that although the PICKit 2 programmer would no longer communicate with either of these PIC 18F252s, the programs that I had already put on them continued to successfully execute - both have copies of the game logic proof of concept code - the difference between the two being that on one of them, the menu button that would start a game with four seeds per pit is programmed to start a game with twenty-four seeds per pit; this was not intended for gameplay, merely as a way to test the pit board power supply under full load.

Not having any PIC18F252s to test the new game logic code I was working on severely effected the progress I was able to make during the Easter holidays, as I had to perform all testing within MPLAB using the MPLAB SIM debugging plug-in. I was able to get some parts of the game logic code written and tested using this, such as a frame buffer and an update to the pit board driver, designed to display the quantity of seeds in a pit as a column of tens and units if the number exceeds twenty-four; necessary for games with a total of more than twenty-four seeds in play (such as when starting a game with three or more seeds per pit).

However I was unable to test parts of the program such as the LCD driver and as a consequence, was unable to start development of the menu systems or the AI player, because this would really have required that I could test the program in the target application, not just a debugger.

At the time of writing this (two days before the deadline), I have just ordered two more PIC18F252s in the hope that they will arrive tomorrow, giving me one day before the deadline in which to try and get as much of game logic code developed (LCD menus & AI player). I am faced with a dilemma because the documentation and practical are due in at the same time, yet I will only know the outcome of my program development attempt moments before the deadline - this begs the question of what to document; the proof of concept program that I wrote, which already works, or the full game logic code including LCD menus and AI player which I hope to develop over the next two days.

Component Usage

Description	*Supplier*	*Rapid Order Number*	*Unit Cost (£)*	*Quantity*	*Total (£)*
PIC16F882-E/SP	Microchip Direct	n/a	0.94	20	18.80
PIC18F252-I/SP	Rapid	73-3302	4.00	2	8.00
PICKit 2 Programmer	Rapid	97-0101	44.00	1	44.00
5mm Ultrabright Red LEDs	Rapid	55-0134	0.09	144	12.96
5mm Ultrabright Blue LEDs	Rapid	55-1662	0.28	144	40.32
5mm Ultrabright White LEDs	Rapid	55-1640	0.30	48	14.40
Bridge Rectifier (3A, 200V)	Rapid	47-3206	0.14	1	0.14
Bridge Rectifier (8A, 100V)	Rapid	47-2894	0.30	2	0.60
L78S05CV +5V 2A Voltage Regulator	Rapid	47-3302	0.44	4	1.76
Square button 12x12mm tactile switches (PCB mounting)	Rapid	78-0640	0.19	16	3.04
Buttons for 12x12mm tactile switches (black, square)	Rapid	78-1185	0.05	16	0.80
Serial LCD module	Rapid	13-1266	14.00	1	14.00
Battery boxes with covers (4x AA)	Rapid	18-2913	0.38	2	0.76
DC Power sockets (2.1mm)	Rapid	20-0878	2.52	1	2.52
Vandal resistant switches (Proud fit 22mm)	Rapid	78-1683	3.65	1	3.65
3-pin Ceramic resonator (10MHz)	Rapid	90-0675	0.25	1	0.25
				Total:	£166.00

Figure 48.0 - Component summary

Component Usage

Money invested in the project

Figure 48.0 shows all components that either a) I had to order specifically or b) are expensive enough to justify inclusion in the table (e.g the serial LCD module - stocked by school but expensive nonetheless). I maintained the table during development for my own reference, and have included it for the sake of documentation completeness as I consider one measure of the quality of documentation to be the ease with which a reader could produce the product described.

At a total cost of £166, not including materials such as fibre glass board for PCBs and acrylic for the case construction, iBantumi is not the cheapest of projects. As I am not the one paying, it is not really my position to pass judgement on whether it was worth the cost, but a serious point in its favour is that having nearly finished the project, had I known at the start the approximate cost and been told that I would have to pay for it, I would definitely do so because I have been able to learn so much from undertaking it.

Game Logic Program Development

	RC0	RC1	RC2	RC3
RC4	Pit 13	Pit 10	Pit 3	Pit 6
RC5	Pit 11	Pit 8	Pit 1	Pit 4
RC6	Pit 12	Pit 9	Pit 2	Pit 5
RB7	Menu 1	Menu 2	Menu 3	Menu 4

Figure 49.0 - Switch array addresses table

After having constructed the case, I swapped around which side of the game logic board the ribbon cables from the pit boards connected to, purely for the sake of tidiness in the case. This of course meant remapping the switch array addresses. Figure 49.0 shows the new addresses.

When creating the game logic code, I created functions for control of each hardware component; one for sending numerical values to the pit LEDs, one for scanning the menu/switch buttons, one for sending 24 bit variables of the 'short long' type (where each bit is an LED) to a pit (used when the numerical value exceeds 24, to show columns of tens and units).

I also created functions for parts of the game logic, such as 'move' and 'AIMove' functions to sow seeds anticlockwise around the board (which is a 15 element array - the first element being the 'hand' and elements 1-14 being pits).

The 'AI' code is incredibly basic (as I only had 2 hours to write it in) and works simply by testing the result of moving each of the six pits belonging to the computer player to see which will result in the biggest gain for the mancala, and then moving with this pit.

In terms of readability and modularity, the code could still be improved. The only justification for this is that I finalised the code in the last 24 hours leading up to the deadline, as the replacement PIC18F252s arrived literally 28 hours before the deadline.

Finally, the version of the program that I actually downloaded to iBantumi contains one bug; when a number of seeds of sufficient number are sown from red's side, around the board such that they pass the mancala and should come back onto red's side, they don't, and instead seem to 'disappear' from the board. I was able to identify the cause of this, and fix it; in the code in this documentation, the bug is fixed, but unfortunately I was already on to the last PIC18F252 that remained programmable at this point, and when I came to download the fixed version, this too had failed with the same symptoms as the others, so I have no programmable PIC18F252s remaining, and thus program development was frozen here.

Figure 49.1 shows the game logic code.

Game Logic Program Development

Figure 49.1 - Game Logic Code

```
#include <p18cxxx.h>
#include <delays.h>
#include <sw_uart.h>

/*************************************************************
        Program:    iBantumi Game Logic Code
        Project:    iBantumi
        Author:             David Piggott
        Date:       30/03/08
*************************************************************/

//    Software UART delay definitions

void  DelayTXBitUART(){
      Delay1KTCYx(4);
      Delay100TCYx(1);
      Delay10TCYx(5);
      Delay1TCY();
      Delay1TCY();
      Delay1TCY();
      Delay1TCY();
      Delay1TCY();
    }
void  DelayRXBitUART(){
    }
void  DelayRXHalfBitUART(){
    }

int       key;
char  activePlayer;
char  activePit;
char  count;
char  bitBoard[15];
char  AISim[15];
char  AIResults[6];
char  AIGain;
char  AIPlayer;
char  AIListen;
char  arrayCopyCount;

//    Hardware driver prototypes

char  keyDriver(void);
char  move(void);
void  piezoDriver(int frequency);
void  LEDNumericDriver(void);
void  LEDFrameDriver(short long frame);
```

Game Logic Program Development

```
//    Hardware driver implementations

char  keyDriver(void){

      //    Scans array addressed keys and returns corresponding
      //    pit number for pit keys or [20-23] for menu keys.
      //    Returns 0 if no key is pressed.

      LATC = 0b00010000;
      switch(0b00001111 & PORTC){
      case 0b00000001:
            return 13;
      break;
      case 0b00000010:
            return 10;
      break;
      case 0b00000100:
            return 3;
      break;
      case 0b00001000:
            return 6;
      break;
      }

      LATC = 0b00100000;
      switch(0b00001111 & PORTC){
      case 0b00000001:
            return 11;
      break;
      case 0b00000010:
            return 8;
      break;
      case 0b00000100:
            return 1;
      break;
      case 0b00001000:
            return 4;
      break;
      }

      LATC = 0b01000000;
      switch(0b00001111 & PORTC){
      case 0b00000001:
            return 12;
      break;
      case 0b00000010:
            return 9;
      break;
      case 0b00000100:
            return 2;
```

Game Logic Program Development

```
    break;
    case 0b00001000:
        return 5;
    break;
    }

    LATC = 0b10000000;
    switch(0b00001111 & PORTC){
    case 0b00000001:
        return 21;
    break;
    case 0b00000010:
        return 22;
    break;
    case 0b00000100:
        return 23;
    break;
    case 0b00001000:
        return 24;
    break;
    default:
        return 0;
    break;
    }
}
void  piezoDriver(int frequency){

    //    Outputs a pulse of frequency frequency with fixed length 500ms

    int   piezoDriverOscCount;

    for(piezoDriverOscCount = 0; piezoDriverOscCount <= frequency / 20; piezoDriverOscCount++){
            PORTBbits.RB0 = 1;
            Delay1KTCYx(20000 / frequency);
            PORTBbits.RB0 = 0;
            Delay1KTCYx(20000 / frequency);
        }
    }
void  LEDNumericDriver(void){

    //    Outputs 14 element array to pit board
    //    showing numeric value either as number of LEDs
    //    lit, or in columns of tens and units.

    int             LEDNumericDriverPitCount;
    int             LEDNumericDriverLEDCount;

    //    Frame definitions
```

Game Logic Program Development

```
short long unitsFrame[9];
short long tensFrame[9];
unitsFrame[0] = 0b00000000000000000000000000000000;
unitsFrame[1] = 0b01000000000000000000000000000000;
unitsFrame[2] = 0b01000100000000000000000000000000;
unitsFrame[3] = 0b01000100010000000000000000000000;
unitsFrame[4] = 0b01000100010001000000000000000000;
unitsFrame[5] = 0b01000100010001000100000000000000;
unitsFrame[6] = 0b11000100010001000100010000000000;
unitsFrame[7] = 0b11001100010001000100010000000000;
unitsFrame[8] = 0b11001100110001000100010000000000;
unitsFrame[9] = 0b11001100110011000100010000000000;
tensFrame[0] = 0b00000000000000000000000000000000;
tensFrame[1] = 0b00010000000000000000000000000000;
tensFrame[2] = 0b00010001000000000000000000000000;
tensFrame[3] = 0b00010001000100000000000000000000;
tensFrame[4] = 0b00010001000100010000000000000000;
tensFrame[5] = 0b00010001000100010001000010000000;
tensFrame[6] = 0b00110001000100010001000010000000;
tensFrame[7] = 0b00110011000100010001000010000000;
tensFrame[8] = 0b00110011001100010001000010000000;
tensFrame[9] = 0b00110011001100110011000010000000;

//     Send start bit
PORTAbits.RA0 = 1;
Delay10TCYx(100);
PORTAbits.RA0 = 0;
Delay10TCYx(25);

    for(LEDNumericDriverPitCount = 1; LEDNumericDriverPitCount <= 14; LEDNumeric-
DriverPitCount++){
            if(bitBoard[LEDNumericDriverPitCount] < 25){
                for(LEDNumericDriverLEDCount = 0; LEDNumericDriverLEDCount <
(24 - bitBoard[LEDNumericDriverPitCount]); LEDNumericDriverLEDCount++){
                        PORTAbits.AN0 = 1;
                        Delay10TCYx(75);
                        PORTAbits.RA0 = 0;
                        Delay10TCYx(75);
                    }
                for(LEDNumericDriverLEDCount = 0; LEDNumericDriverLEDCount <
bitBoard[LEDNumericDriverPitCount]; LEDNumericDriverLEDCount++){
                        PORTAbits.AN0 = 1;
                        Delay10TCYx(50);
                        PORTAbits.RA0 = 0;
                        Delay10TCYx(75);
                    }
            }else{
                LEDFrameDriver(tensFrame[bitBoard[LEDNumericDriverPitCount] /
10] | unitsFrame[bitBoard[LEDNumericDriverPitCount] % 10]);
                }
            }
```

Game Logic Program Development

```
}

void  LEDFrameDriver(short long frame){

      //    Takes a 24 bit variable (short long) where each bit
      //    corresponds to a pit LED, and outputs them to the
      //    pit boards. The pit boards are not reset before hand,
      //    so which LEDs are switched depends at what position in
      //    output frame this is called. frame is read from right
      //    to left, so the least significant bit is the top left
      //    LED.

      int
      short long    LEDFrameDriverLEDCount;
                    LEDFrameDriverBitMask = 0b00000000000000000000000001;

      for(LEDFrameDriverLEDCount = 1; LEDFrameDriverLEDCount <= 24; LEDFrame-
DriverLEDCount++){
            if((frame & LEDFrameDriverBitMask) != 0){
                  PORTAbits.AN0 = 1;
                  Delay10TCYx(50);
                  PORTAbits.RA0 = 0;
                  Delay10TCYx(75);
            }else{
                  PORTAbits.AN0 = 1;
                  Delay10TCYx(75);
                  PORTAbits.RA0 = 0;
                  Delay10TCYx(75);
            }
            LEDFrameDriverBitMask = LEDFrameDriverBitMask << 1;
      }
}

char  move(void){
      if(bitBoard[0] != 0){
            activePit++;
            if(activePit == 15){
                  activePit = 1;
            }else if(activePit == 7 && activePlayer == 1){
                  activePit++;
            }else if(activePit == 14 && activePlayer == 0){
                  activePit = 1;
            }
            bitBoard[activePit]++;
            bitBoard[0]--;
            return 1;
      }else{
            return 0;
      }
}

char  AIMove(void){
```

Game Logic Program Development

```
if(AISim[0] != 0){
        activePit++;
        if(activePit == 15){
                activePit = 1;
        }else if(activePit == 7 && activePlayer == 1){
                activePit++;
        }else if(activePit == 14 && activePlayer == 0){
                activePit = 1;
        }
        AISim[activePit]++;
        AISim[0]--;
        return 1;
}else{
        return 0;
        }
}
void  winner(void){
    Delay10KTCYx(0);
    key = 0;
    while(key == 0){
        if(bitBoard[7] > bitBoard[14]){
                PORTAbits.RA0 = 1;
                Delay10TCYx(100);
                PORTAbits.RA0 = 0;
                Delay10TCYx(25);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0101010101010101010101010101);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                LEDFrameDriver(0b0000000000000000000000000000);
                piezoDriver(4000);
                Delay10KTCYx(50);
                PORTAbits.RA0 = 1;
                Delay10TCYx(100);
                PORTAbits.RA0 = 0;
                Delay10TCYx(25);
                LEDFrameDriver(0b1010101010101010101010101010);
                LEDFrameDriver(0b1010101010101010101010101010);
                LEDFrameDriver(0b1010101010101010101010101010);
                LEDFrameDriver(0b1010101010101010101010101010);
                LEDFrameDriver(0b1010101010101010101010101010);
                LEDFrameDriver(0b1010101010101010101010101010);
```

Game Logic Program Development

```
    LEDFrameDriver(0b10101010101010101010101010101010);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    LEDFrameDriver(0b00000000000000000000000000000000);
    piezoDriver(5000);
    Delay10KTCYx(50);
  }else if(bitBoard[14] > bitBoard[7]){
      PORTAbits.RA0 = 1;
      Delay10TCYx(100);
      PORTAbits.RA0 = 0;
      Delay10TCYx(25);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b01010101010101010101010101010101);
      LEDFrameDriver(0b01010101010101010101010101010101);
      LEDFrameDriver(0b01010101010101010101010101010101);
      LEDFrameDriver(0b01010101010101010101010101010101);
      LEDFrameDriver(0b01010101010101010101010101010101);
      piezoDriver(4000);
      Delay10KTCYx(50);
      PORTAbits.RA0 = 1;
      Delay10TCYx(100);
      PORTAbits.RA0 = 0;
      Delay10TCYx(25);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b00000000000000000000000000000000);
      LEDFrameDriver(0b10101010101010101010101010101010);
      LEDFrameDriver(0b10101010101010101010101010101010);
      LEDFrameDriver(0b10101010101010101010101010101010);
      LEDFrameDriver(0b10101010101010101010101010101010);
      LEDFrameDriver(0b10101010101010101010101010101010);
      LEDFrameDriver(0b00000000000000000000000000000000);
      piezoDriver(5000);
```

Game Logic Program Development

```
        Delay10KTCYx(50);
    }
    key = keyDriver();
  }
}
void  main(void){

  //            Initialise I/O
  LATA = 0;
  LATB = 0;
  LATB = 0;
  ADCON1 = 0b00000110;
  TRISA = 0b00000000;
  TRISB = 0b00010110;
  TRISC = 0b00001111;
  //OpenUART();
  PORTBbits.RB5 = 1;

  //          Initialise board with two seeds per pit, excluding mancala
  for(count = 1; count <= 14; count++){
      bitBoard[count] = 2;
  }
  AIPlayer = 0;
  bitBoard[7] = 0;
  bitBoard[14] = 0;
  LEDNumericDriver();
  activePlayer = 0;

  //          Begin gameplay
  while(1){
      if(activePlayer == 1 && AIPlayer == 1){
          Delay10KTCYx(0);
          Delay10KTCYx(0);
          piezoDriver(4000);
          Delay10KTCYx(25);
          piezoDriver(4000);
          for(count = 8; count < 14; count++){
              for(arrayCopyCount = 0; arrayCopyCount < 15; arrayCopy-
Count++){
                  AISim[arrayCopyCount] = bitBoard[arrayCopyCount];
              }
              AISim[0] = AISim[count];
              AISim[count] = 0;
              activePit = count;
              //      Move the seeds
              while(AIMove()){
              }
              //      Was the turn eligable for capture
              if((AISim[activePit] == 1) && (8 <= activePit && activePit <= 13))
  {
```

Game Logic Program Development

```
//      Move the opposite seeds to the active players
mancala
                AISim[14] += AISim[14 - activePit];
                AISim[14 - activePit] = 0;
//      Move the capture trigger pit seeds into the man-
cala
                AISim[14] += AISim[activePit];
                AISim[activePit] = 0;
            }
            AIResults[count] = AISim[14] - bitBoard[14];
        }
        AIGain = 0;
        key = 13;
        for(count = 8; count < 14; count++){
            if(AIResults[count] > AIGain){
                AIGain = AIResults[count - 8];
                key = count;
            }
        }
        bitBoard[0] = bitBoard[key];
        bitBoard[key] = 0;
        activePit = key;
//      Move the seeds
        while(move()){
            Delay10KTCYx(200);
            LEDNumericDriver();
            piezoDriver(2000);
        }
//      Move complete, evaluate stop conditions
//      Was the turn eligable for capture
        if((bitBoard[activePit] == 1) && (8 <= activePit && activePit <= 13)){
//          Move the opposite seeds to the active players mancala
and update the board
            bitBoard[14] += bitBoard[14 - activePit];
            bitBoard[14 - activePit] = 0;
            Delay10KTCYx(200);
            LEDNumericDriver();
            piezoDriver(3000);
//          Move the capture trigger pit seeds into the mancala and
update the board
            bitBoard[14] += bitBoard[activePit];
            bitBoard[activePit] = 0;
            Delay10KTCYx(200);
            LEDNumericDriver();
            piezoDriver(4000);
//          Hand play over to the opposition
            activePlayer = 0;
//          Was the last seed sown into the player's own mancala?
        }else if(activePit == 14){
//          Grant another move
        }else{
```

Game Logic Program Development

```
//      Hand over play to the opposition
        activePlayer = 0;
    }
}
key = 0;
while(key == 0){
    if(PORTBbits.RB1 == 1){
        PORTBbits.RB5 = 1 - PORTBbits.RB5;
        Delay10KTCYx(0);
        LEDNumericDriver();
        piezoDriver(2000);
    }
    if(bitBoard[1]+bitBoard[2]+bitBoard[3]+bitBoard[4]+bitBoard[5]
+bitBoard[6] == 0){
        bitBoard[14] += bitBoard[8]+bitBoard[9]+bitBoard[10]+bitBoard
[11]+bitBoard[12]+bitBoard[13];
            bitBoard[8] = 0;
            bitBoard[9] = 0;
            bitBoard[10] = 0;
            bitBoard[11] = 0;
            bitBoard[12] = 0;
            bitBoard[13] = 0;
            winner();
        }else if(bitBoard[8]+bitBoard[9]+bitBoard[10]+bitBoard[11]+bitBoard
[12]+bitBoard[13] == 0){
            bitBoard[14] += bitBoard[8]+bitBoard[9]+bitBoard[10]+bitBoard
[11]+bitBoard[12]+bitBoard[13];
            bitBoard[8] = 0;
            bitBoard[9] = 0;
            bitBoard[10] = 0;
            bitBoard[11] = 0;
            bitBoard[12] = 0;
            bitBoard[13] = 0;
            winner();
        }
        key = keyDriver();
        if(activePlayer == 1 && AIPlayer == 1){
            key = 30;
        }
    }
    if(key < 20){
        AIListen = 0;
    }
//                  Wait for player to initialise their move
    if((1 <= key && key <= 6 && activePlayer == 0) | (8 <= key && key <= 13 &&
activePlayer == 1)){
//      Move contents of pit into off board 'hand'
        bitBoard[0] = bitBoard[key];
```

Game Logic Program Development

```
                bitBoard[key] = 0;
                activePit = key;
                //      Move the seeds
                while(move()){
                        Delay10KTCYx(200);
                        LEDNumericDriver();
                        piezoDriver(2000);
                }
                //      Move complete, evaluate stop conditions
                //      Was it player one's turn and was the turn eligable for
capture
                if((bitBoard[activePit] == 1) && (activePlayer == 0) && (1 <=
activePit && activePit <= 6)){
                        //      Move the opposite seeds to the active players
mancala and update the board
                                bitBoard[7] += bitBoard[14 - activePit];
                                bitBoard[14 - activePit] = 0;
                                Delay10KTCYx(200);
                                LEDNumericDriver();
                                piezoDriver(3000);
                                //      Move the capture trigger pit seeds into the man-
cala and update the board
                                bitBoard[7] += bitBoard[activePit];
                                bitBoard[activePit] = 0;
                                Delay10KTCYx(200);
                                LEDNumericDriver();
                                piezoDriver(4000);
                                //      Hand play over to the opposition
                                activePlayer = 1 - activePlayer;
                        //      Was it player two's turn and was the turn eligable for
capture
                }else if((bitBoard[activePit] == 1) && (activePlayer == 1) && (8
<= activePit && activePit <= 13)){
                        //      Move the opposite seeds to the active players
mancala and update the board
                                bitBoard[14] += bitBoard[14 - activePit];
                                bitBoard[14 - activePit] = 0;
                                Delay10KTCYx(200);
                                LEDNumericDriver();
                                piezoDriver(3000);
                                //      Move the capture trigger pit seeds into the man-
cala and update the board
                                bitBoard[14] += bitBoard[activePit];
                                bitBoard[activePit] = 0;
                                Delay10KTCYx(200);
                                LEDNumericDriver();
                                piezoDriver(4000);
                                //      Hand play over to the opposition
                                activePlayer = 1 - activePlayer;
                                //      Was the last seed sown into the player's own man-
cala?
```

Game Logic Program Development

```
    }else if((activePit == 7 && activePlayer == 0) | (activePit == 14
&& activePlayer == 1)){
                //      Grant another move
            }else{
                //      Hand over play to the opposition
                activePlayer = 1 - activePlayer;
            }
        }else if(key == 21 && AIListen == 0){
            //          Restart the game with one seeds per pit, excluding man-
cala
            for(count = 1; count < 15; count++){
                bitBoard[count] = 2;
            }
            AIPlayer = 0;
            bitBoard[7] = 0;
            bitBoard[14] = 0;
            LEDNumericDriver();
            piezoDriver(2000);
            activePlayer = 0;
            AIListen = 1;
            Delay10KTCYx(100);
        }else if(key == 22 && AIListen == 0){
            //          Restart the game with seeds per pit, excluding mancala
            for(count = 1; count < 15; count++){
                bitBoard[count] = 3;
            }
            AIPlayer = 0;
            bitBoard[7] = 0;
            bitBoard[14] = 0;
            LEDNumericDriver();
            piezoDriver(2000);
            activePlayer = 0;
            AIListen = 1;
            Delay10KTCYx(100);
        }else if(key == 23 && AIListen == 0){
            //          Restart the game with three seeds per pit, excluding
mancala
            for(count = 1; count < 15; count++){
                bitBoard[count] = 4;
            }
            AIPlayer = 0;
            bitBoard[7] = 0;
            bitBoard[14] = 0;
            LEDNumericDriver();
            piezoDriver(2000);
            activePlayer = 0;
            AIListen = 1;
            Delay10KTCYx(100);
        }else if(key == 24 && AIListen == 0){
            //          Restart the game with four seeds per pit, excluding man-
cala
```

Game Logic Program Development

```
for(count = 1; count < 15; count++){
        bitBoard[count] = 5;
    }
    AIPlayer = 0;
    bitBoard[7] = 0;
    bitBoard[14] = 0;
    LEDNumericDriver();
    piezoDriver(2000);
    activePlayer = 0;
    AIListen = 1;
    Delay10KTCYx(100);
}else if(key == 21 && AIListen == 1){
    //          Restart the game with one seeds per pit, excluding man-
cala
    for(count = 1; count < 15; count++){
        bitBoard[count] = 2;
    }
    AIPlayer = 1;
    bitBoard[7] = 0;
    bitBoard[14] = 0;
    LEDNumericDriver();
    piezoDriver(2000);
    activePlayer = 0;
    AIListen = 0;
    Delay10KTCYx(100);
}else if(key == 22 && AIListen == 1){
    //          Restart the game with seeds per pit, excluding mancala
    for(count = 1; count < 15; count++){
        bitBoard[count] = 3;
    }
    AIPlayer = 1;
    bitBoard[7] = 0;
    bitBoard[14] = 0;
    LEDNumericDriver();
    piezoDriver(2000);
    activePlayer = 0;
    AIListen = 0;
    Delay10KTCYx(100);
}else if(key == 23 && AIListen == 1){
    //          Restart the game with three seeds per pit, excluding
mancala
    for(count = 1; count < 15; count++){
        bitBoard[count] = 4;
    }
    AIPlayer = 1;
    bitBoard[7] = 0;
    bitBoard[14] = 0;
    LEDNumericDriver();
    piezoDriver(2000);
    activePlayer = 0;
    AIListen = 0;
```

Game Logic Program Development

```
        Delay10KTCYx(100);
    }else if(key == 24 && AIListen == 1){
        //          Restart the game with four seeds per pit, excluding man-
cala
        for(count = 1; count < 15; count++){
            bitBoard[count] = 5;
        }
        AIPlayer = 1;
        bitBoard[7] = 0;
        bitBoard[14] = 0;
        LEDNumericDriver();
        piezoDriver(2000);
        activePlayer = 0;
        AIListen = 0;
        Delay10KTCYx(100);
    }
}
}
```

Testing

In terms of gameplay, iBantumi has been through continual play testing from the moment that I got the first game logic proof of concept programming. Fortunately I had no shortage of volunteers to play the game, and the testing proved very useful to identify flaws in the implementation of the game rules. For example, one area in which the implementation was initially incorrect was the way in which the rules of capture were implemented; I had incorrectly programmed it so that if the last seed of a turn was sown to an empty pit belonging to the player who started the turn, but the pit opposite it was empty (i.e. no seeds available for capture), capture would not take place (i.e. the last seed sown by that player would remain in the pit, rather than being moved into the mancala). This turned out to be an incorrect implementation, despite me having gone to extra effort to implement it, though on the plus side this meant it was simple enough to fix.

Other mistakes/design flaws that I noticed in testing are identified and detailed in the evaluation against specification included on previous pages, and the “Negative points not covered by evaluation against specification” - one particular area of weakness being the power supply. The testing which lead to my theory of the mechanism by which the red PORTC LEDs fail to light is described in the section “Pit Boards Smoke Test”.

In terms of case testing, I have already identified strengths and weaknesses in the design and construction on the previous pages. However one way by which I tested it has yet to be documented, and was not intentional. Several weeks ago, as I had just finished case construction, I had taken iBantumi home to work on as usual. iBantumi was on my desk, and the wooden Bao board shown in figure 1.0 was on a bookshelf, approximately one metre directly above iBantumi and next to a file - I was reaching up to get the file, and as I pulled it off the shelf, it caught the wooden Bao board which was pulled off the shelf with it and fell directly onto iBantumi below. I estimate the mass of the Bao board to be 0.5kg, so it was no an insignificant momentum with which it collided with iBantumi, and despite this, iBantumi was completely unscathed by the incident.

The second means by which it has been tested was when, in at attempt to remedy the downward bowing of the lower panel, I put several large masses (two laptops, a Rapid catalogue, and several plastic trays) directly on top of the case. While this hasn’t fixed the downward bowing of the lower panel, it did prove that the case is of more than sufficient strength and integrity to allow it to be handled by users without the risk of it breaking.

Evaluation

Evaluation Against Specification

Below follows my original specification and my evaluation of each point.

Player Input - Specification Points

1. Each pit will have a push button switch, pressed by the player to initiate their move by taking the seeds from that pit into the hand.
2. There will be one LCD, shared between the two players, positioned at the end of the board such that it can be seen by both players.
3. The LCD menus will be controlled by four push button switches; two to change the selection, and two to cancel/confirm the selection.
4. This totals $12_{pit} + 4_{menu} = 16_{total}$ switches, which can be arranged and scanned in a 4x4 array, such that only 8 I/O pins (one port) are required on the main controlling PICmicro.

Player Input - Evaluation

1. There is a push button switch for each pit. In terms of functionality they are ideal, being 12mm square provides plenty of surface area, and as they are tactile the player receives feedback to confirm they have fully depressed the switch. The scanning of these switches, which are connected in an array setup, is flawless and the end user would not be able to tell the difference between this setup and a direct connection - the only situation in which problems might arise is if there were a requirement for several switches to be depressed simultaneously - but as this is not required there is no problem. In terms of cost, all 16 cost £3.84 which is quite reasonable for a prototype.
2. There is one LCD shared between the two players and it is positioned at the end of the board. However this position is not optimal for visibility to both players. It would be more easily visible if it was angled at approx 45 degrees as I had shown it on my initial casing ideas. However I have two reasons for not having done this
   a) the extra space required by the LCD module as a result of mounting it at 45 degrees would have either required the case to be several centimetres taller, or would have required that the LCD protruded above the top of the case - but I had decided to avoid doing this as it would be a liable to damage in this position.
   b) b) as it was not possible to use CAMM to produce the upper panel for the case, this would have just meant introducing more complexity to the case design when I had limited time in which to produce it and so could not afford to make mistakes.
3. Due to a combination of poor planning when creating the game logic artwork, a lack of information available from Microchip on the software USART library functions, and the multiple week delay in programming caused by the failure of the original PIC18F252s I had to pro-

Evaluation

gram, I have been unable to control the LCD, despite maintaining the problem solving effort up until the deadline. However there is of coure nothing wrong with the menu buttons and I have been able to develop a control system using these that works without the LCD, though it is of course limited in terms of what can be done without graphical/textual information. Despite this, games can be started with a range of seed quantities, and the option of human vs. human or huma vs. computer gameplay.

4. The switches are arranged in a 4x4 array and only one port is used for scanning them. This allowed a relatively small pincount on the PIC.

Case - Specification Points

1. The case will be based on case idea six.
2. The main material will be transparent acrylic. Some parts may be coloured translucent acrylic (e.g. red/blue for the mancala ends to indicate possession of the seeds in that area by the player).
3. The on/off switch will be a slide switch mounted on the same side of the case as the LCD.

Case - Evaluation

1. The case is based on idea six, though the initial sketch was very open to interpretation. The key differences between the initial ideas was the LCD quantity and placement, and the quantity and placement match those in idea six with the exception of the angle, already discussed on the previous page.
2. The entire construction is transparent acrylic. During the case design stages I considered the use of aluminium to create the side panels, but decided that aesthetically this wouldn't be as good due to the aluminium tabs interfering with the clear view through the top panel. Additionally, it would have been harder to work with and thus the finish would have been of lower quality than that which I have achieved with acrylic. I decided not to use coloured acrylic for any parts because this inconsistency would have lowered the aesthetic quality.
3. The on/off switch is not in fact a slide switch; instead, I opted for a push button switch simply for aesthetic reasons; I could not find any slide switches that were consistent with the metal and plastic design of the case. It is however at the same end as the LCD, as are all I/O functions, which helps keep the remaining three sides of the case uniform.

LED Control - Specification Points

1. The 336 seed LEDs will be grouped into arrays of 24 LEDs, and each group controlled by a 28 pin PICmicro, programmed with assembly code, that I will refer to as LED registers.

Evaluation

2. Each LED will be connected directly to an output on an LED register.
3. The LED registers will be controlled by a serial pulse train from the main controlling PICmicro.
4. Therefore the LED registers will have a minimum of 25 I/O pins (24 output, 1 input for the serial pulse train).
5. The control protocol will be detailed in the development section, as and when I develop it.

LED Control - Evaluation

1. The 336 seed LEDs are grouped into arrays of 24 LEDs, and each group is controlled by a 28 pin PICmicro, programmed in assembly code.
2. Each LED is directly connected to an output on an LED register. This approach was successful with the blue LEDs but not for all the red LEDs, as detailed in the section titled “Pit Boards Smoke Test”. The cause of this is of course because I didn’t put series current limiting resistors in; it was desirable to avoid them at all costs as including them would have significantly increased the artwork sizes, such that they wouldn’t even fit on A4 paper, and thus wouldn’t fit the exposure box, developer tray or etch tank, aswell as creating another 672 holes to drill and joints to solder.

   However, I didn’t just decide to not include current limiting resistors without some calculation; the LEDs that I had selected following my component research have a maximum current rating of 30mA, while the maximum current rating of the PIC16F882 pins is 25mA. From my experience with PICs, I knew that the current would be limited by the maximum output of the PIC to 25mA - below that of the LEDs, and thus that they would operate correctly without current limiting resistors.

   While I am not entirely certain of the workings of the problem that exists with the red LEDs connected to PORTC pins, my theory is that when the voltage across the PIC16F882s is high enough that even when the drop due to switching is taken into account, the p.d. across the LEDs connected to PORTC is so high that the current passing through these LEDs is so high that a safety brown-out feature of PORTC pins is triggered (I have not heard/read of one - this is merely speculation on my part) causing them to switch off.

   This theory is consistent with the fact that if all 288 red and blue LEDs are switch on and left on for several minutes, the voltage regulators temperatures increase such that their increased resistances cause a voltage drop that is sufficient to lower the p.d. across the PORTC LEDs by enough that the current flowing in them no longer triggers this safety brown-out.

Evaluation

It is also consistent with the fact that when all 288 red and blue LEDs are switched on the red PORTC LEDs can be seen momentarily pulsing on before switching off, and with the fact that when the voltage regulators have warmed up enough, they do not spontaneously switch on; a new packet must be transmitted to the pit registers reinstructing them to switch on. A possible explanation for why this doesn't occur with red PORTA or PORTB LEDs is that the brown-out level of these is different. The reason the problem doesn't occur with blue PORTC LEDs is that the blue LEDs have a higher resistance and so draw less current for a given p.d.
With hindsight, it wouldn't actually have been that much extra work to include current limiting resistors, as they wouldn't have needed to be on the PCBs themselves; I could have used them in place of the ribbon cable interlinking the boards. This would of course have required large amounts of heatshrink in order to insulate between them, and would have introduced the risk of 'dead pixels' by means of metal fatigue in the resistors causing them to break. However the effect of the red PORTC LEDs not working was not so bad that I deemed retro-fitting current limiting resistors worth the time it would have taken (probably a whole week), as it would have delayed other crucial parts of the project.

3. The LED registers are controlled by a serial pulse train from the main controlling PICmicro, and despite being asynchronous I have yet to observe a single transmission error; that is to say, the pulse lengths I chose are not too short or too similar, while still allowing a mini mum frame rate of 29Hz!
4. The LED registers have 25 I/O pins (24 output, 1 input for the serial transmission).
5. The control protocol is detailed in the development section, and I believe it is the simplest and most efficient protocol possible to achieve what it does; efficient in terms of the amount of useful data transmitted compared with the total information transmitted, and simple in that it is relatively easy to understand the protocol itself.

Processing - Specification Points

1. The rules described in the following section (gameplay) will be applied by a single PICmicro, programmed using C.
2. One output pin will be required for the piezo sounder.
3. One input pin will be required for the on/off switch.
4. Eight I/O (four input, four output) will be required for the push button switches.
5. One output pin will be required for transmitting the serial pulse train to the LED registers.
6. One to eight output pins will be required for controlling the LCD, depending on how I choose to drive it (whether the driver is controlled by serial or parallel signalling).

Evaluation

7. Therefore a maximum of (1 + 1 + 8 + 1 + 8) = 19 I/O pins will be required on the controlling PICmicro.

Processing - Evaluation

1. While the rules of the game are correctly applied by the game logic controller, because the program was only finalised in the 24 hours leading up to the deadline (see section titled “Time Usage & Unfortunate Circumstances”), I not satisfied with the program in terms of readability and modularity; it was very much a mad panic. If it weren’t for the PIC18F252s failing to program in the Easter holidays, it is likely that I would have been able to get the LCD menus working, along with a more advanced and challenging AI player, and I would have been able to take my time when programming to ensure maximum modularity and readability.
2. One output is used for the piezo sounder.
3. One input is used for the on/off button.
4. Eight I/O are used for the pit buttons and menu buttons.
5. One output pin is used for transmitting the serial pulse train to the LED registers. However an additional output pin is used to control the relay that controls power to the pit boards.
6. I had planned to use two pins to control the serial LCD module - one for the serial transmission, and one to actually power the module, as the datasheet for it stated a current draw of less than 5mA. The reason for doing this, rather than having it permanently connected to +5V was because of the software power management; when in standby it is desirable to minimise power consumption in order to maximise battery life. However, when I tried testing the LCD in clock mode (to isolate other causes) via the PIC18F252, it simple didn’t function, and the p.d. measured across it was 0v leading me to suspect the output pin of the PIC18F252 was browning out as with the red PORTC LEDs on the register PICs. Fortunately however, the track that went to this pin was only 0.254cm from a +5V track so I was able to bridge the gap with solder, and set the tristage bit for that pin to 1 - high impedance (input). This solved the power issue and the LCD powered up correctly.
7. 13 I/O pins are used on the PIC18F252.

Gameplay - Specification Points

1. The objective is to have the most seeds in your mancala at the end of play.
2. There will be six pits per player.
3. There will be one mancala per player, situated to the right of their six pits.

Evaluation

4. Gameplay will start with two seeds in each pit.
5. A turn consists of one or more moves.
6. Whether another move is granted depends on the outcome of the previous move.
7. A move starts when a player takes the seeds from one pit into the hand (by pressing the switch located in that pit).
8. The seeds are then sown anticlockwise one at a time around the board.
9. Seeds cannot be moved from the mancala.
10. If when a move terminates, the last seed in the hand was sown into the player's mancala, the player is granted (and is obliged to take) another move.
11. If when a move terminates, the last seed in the hand was sown into a pit directly opposing an occupied opponents pit, the seeds in that pit and the opponents pit are moved at once into the mancala of the player who initiated the move.
12. The game terminates when either player has no seeds left in play, and the winner is the player with the most seeds on their side of the board.

Gameplay - Evaluation

1. The rules of the game are applied correctly at all points in the game. I have exceeded the specification in this area, as the game doesn't have to start with two seeds; three, four or five are also available.

Extra Features - Specification Points

- The AI player implementation is subject to be having the time and knowledge/ability to do so.
- I may also add, accessible via the LCD menu, some "Easter eggs"; i.e. the ability to strobe all the LEDs for use as a floodlight type device.

Extra Features - Evaluation

1. As I wrote the final program within the last 24 hours leading up to the deadline, I had considerably less time available for this; nevertheless, I wanted to give it a go and so despite not having LCD menus to access it by, I came up with the double button tap method, and created a very very basic computer player - considering I wrote and debugged this in about two hours, I am very pleased with it.
2. As I was unable to get the serial control of the LCD working, there are of course no LCD menus and so no way to make these 'Easter eggs' available. However, the presence or lack of such features is insignificant in the greater scheme of things.

Evaluation

Negative points not covered by evaluation against specification

The power supply and management is an area of the project that I seriously overlooked, and this has caused several problems. The problems can be summarised as follows:

- High running temperature of pit board voltage regulators due to lack of heatsinks (by chance this is beneficial as it allows the red PORTC LEDs to work once the regulators warm up and voltage drop is sufficient).
- Design fault with LCD power supply required modification of PCB and resulted in increased standby current.
- Generally high standby current results in very short battery life.
- Use of relay to control power to pit boards increases operating current to be higher than necessary.
- Excessive use of bridge rectifiers to protect against reverse polarity was unnecessary and leads to inefficiency due to the voltage drop across them.
- Insufficient planning of connection of batteries results in them being used to power the board even when external power supply is connected as they are connected in parallel and there the batteries are not disconnected, mechanically or electronically when external power is connected. This means that battery run time decreases even running from external power.
- I had to add an inline diode in one wire from the batteries in order to prevent the power supply from charging them (not desirable with non-rechargeables!) as I had failed to realise this would occur when creating the game logic artwork.

In retrospect a simpler and superior power system would have been to have only a 6V rail throughout the product, with no bridge rectifiers or diodes losing energy, and no voltage regulators. It would be powered by a 6V DC PSU and 6V battery (4x AA cells). Because there would be no power processing (bridge rectifiers or voltage regulators) on the pit boards, a relay would no longer be necessary for controlling the pitboards (it was only required as they are bipolar if used in a DPDT setup) and instead I could have used a MOSFET - smaller, more reliable, silent, cleaner (less noise), and without the requirement of a Darlington driver. The small voltage drop across it would not matter as it would simply bring the p.d. across the pit boards closer to 5V.

A system would be required that would switch to the external power supply when present and back to batteries when removed. The simplest way of doing this would be to take advantage of the third connection on the DC power socket I used that acts as a normally closed switch, opening when the power jack is inserted. A large capacitor on the game logic board would maintain sufficient voltage across the game logic

Evaluation

PIC to avoid a reset during the transition. However, one drawback of this would be that if the power jack were present but it wasn't plugged into the mains, there would be no power at all and so reset would occur. An alternative system would be one in which a relay was used controlling which power supply is selected (batteries or external) with the default position being batteries; the external power supply would be connected to the relay coil so that when external power is connected, the relay would switch to disconnect the batteries and connect external power. However a drawback of this system would be the current drawn by the relay coil, which would reduce standby and running times - perhaps a similar system could be implemented using one or more MOSFETs.

LCD

This too is a significant failing of the project, which cannot be attributed to any specific fault but is better described as a series of unfortunate events; it started when I created the game logic artworks and failed to consider that the PIC18F252 has built in hardware USART and I2C controllers; had I realised at the time, I would have tracked the LCD to one of these. However, being preoccupied with other more urgent problems at the time, I overlooked this and tracked it to a standard I/O pin, intended to research how to actually control the LCD later on. When I did, I realised I would have been better off using hardware USART/I2C, but also found that there is a software UART implementation in the C18 code libraries. However, in order to use these library functions I had to define a delay function (as this is dependent on the required baud and the oscillator frequency of the microcontroller). Having done this and recompiled the library, the LCD didn't work; for example sending a 'a' character to it would result in a 'p' showing on the LCD, or sending a 'b' would results in a '_'. Having tried everything I could, and unable to find assistance with the issue, with only twelve hours left until the deadline I decided my time would be better spent on other things. I believe the root of the issue was either poor synchronisation between the PIC18F252 and serial driver, or incorrect timing.

While I am happy with the case design (with the exception of the LCD position as already described), the problem of not being able to fit the panels in the CAMM machine and me subsequently having to produce everything by hand, combined with the tight timescale has resulted in some noticeable inaccuracies, for example:

- Dents on the edge of one end of the lower panel, created when it was bandsawn and overlooked by me during filing and polishing (I did manage to remove them from all other edges though).
- Inconsistency of dimensions and angles of the edges of the tabs joining side panels to upper panels.
- Inaccuracy in position of holes for LCD result in it not being aligned with the menu buttons or edges of the upper panel.

Evaluation

- Inaccuracy in position of holes for pit board standoffs results in tension in the upper pit board which results in the case being bowed downwards in the centre.
- Inaccuracy in cutting of upper panel has resulted in it not being square with the lower panel, and as a result neither are the side panels (relative to the lower panel) as they are attached squarely to the upper panel.
- Lack of support by means of standoffs in the centres of the pit boards has resulted in them being very flexible and liable to displacement when central pit buttons are depressed.

Positive points not covered by evaluation against specification

Positive points not covered by the evaluation of the specification are generally concerning the project as opposed to the product. While I have been able to identify various areas of weakness in the product, it is important not to forget that it was an ambitious project to take on.

On the whole I consider the project a great success; the majority of my work has been outside of the A2 syllabus and so not only have I had to do a lot of development work, but I had to teach myself to program in both PIC assembly and C, as well as identify and familiarise myself with Microchip's development systems - MPLAB IDE and the PICKit 2. This I succeeded in doing, and so was able to successfully develop register code for the PIC16F882s in PIC assembly and game logic code for the PIC18F252 in C.

While I do not consider myself to have mastered either language by means (I did not expect to - the project was not merely about programming - there were also large amounts of circuit and case development and production, and I was doing it along side two other A-Levels), I do not consider this a failing. I am happy to have introduced myself to the languages and gained some experience with them, particularly assembly code because it is such a low level language (I had already used languages with similar syntax to C prior to the project).

Another merit of the project is that part way through development, I realised that a lot of development tasks depended on others in a nearly circular fashion, so I had to revise my strategy and take the only, albeit risky, option of producing artworks and PCBs having done next to no breadboarding. This was most risky for the pit boards due to the amount of work involved in producing them and yet by means of stringent error checking and an entirely theoretical (various artworks designed but no PCBs produced initially) evolutionary process, I was able to create a design that worked first time, albeit it with minor issues such as the red PORTC LEDs already mentioned (this wasn't detected when I created

Evaluation

the initial register prototype board because I didn't populate it with red LEDs on PORTC). The same was not true of the game logic board development, which had to go through a practical evolutionary process but this was not a problem as it has a much smaller component count. Although not an achievement in terms of ingenuity, actually producing and populating pit boards with 336 LEDs was quite satisfying, albeit a lot of work.

A final strength of the project, and something I am very pleased with is the compactness of the case that I was able to achieve by exact modelling of all components in ProDESKTOP. This allowed me to maximise use of free space inside the case while ensuring no physical interference between components.

Industrial Practices, Social Issues, Systems & Control

Before considering how my product would be produced commercially, it is important to first of all consider the feasibility of the design for mass production. As I have only produced a prototype, I didn't have to consider this, which has resulted in a fairly expensive product in terms of component cost and the time taken for me to manufacture it. The most outstanding materials costs are the LEDs, PIC16F882s, and serial LCD module. The cost of the serial LCD module could be reduced significantly by replacing it with a parallel driven LCD (a large part of the price on serial LCD modules is the serial controller PIC and associated circuitry, as well as the royalty on the program that runs on it). This leaves the cost of the LEDs and PIC16F822s outstanding, which as parts of the system are tied closely to each other.

The pitboards are not only expensive in terms of money, but also assembly time and the space required in the case. The best way of decreasing all these expenses would be to completely redesign them as multi-layer boards (this would reduce materials costs, space required, and assembly time), and replace the LEDs and PICs with surface mount devices (including surface mount current limiting resistors) - again reducing materials costs, space required, and assembly time, when considered in the context of mass production. The LEDs would be on the upper side of the board, and the PICs on the lower side. Because this would allow the boards to be smaller, and because the size wouldn't be limited like it was with the PCB production equipment at school, both player one and player two pit boards could be produced as one board.

In fact, the game logic board could also be merged with the pit boards so that the entire design would consist of a single multi-layer PCB. Because the product would no longer be a prototype, the ICSP programming connectivity could be removed from the PCB and case, simplifying the design, reducing component costs, and saving space. Additionally, the vandal resistant power button is one component of disproportionate cost, and this would be replaced with a simple slide switch of similar style to the hold switches found on MP3 players - which is appropriate because it's function would be similar - disabling the pit and menu buttons, to prevent accidental input, but also switching off the pitboards to save power.

Because the power supply would be completely redesigned based on my ideas described in the evaluation (6V supply throughout, with solid state intelligent switching - i.e. no relays), only 4x AA batteries would be required and so only one battery box would be needed in the case - again reducing costs, assembly time, and space required. This wouldn't impact on the battery life at all - infact, it would be increased due to the far superior power supply and management design, as well as the reduced current drawn by surface mount LEDs.

Industrial Practices, Social Issues, Systems & Control

The case itself too would have to be reconsidered - while producing the panels with CAMM would be cheaper and quicker per unit than manually producing them with a hegner saw etc. it would be cheaper and quicker still to produce the case by either injection moulding it, in two halves - upper and lower, because the unit cost savings would far outweigh the higher initial investment in production setup.

Combining all these improvements, approximate dimensions of 10cm width x 25cm length x 3cm thickness are not unreasonable estimates - with the most significant occupier of space now being the single 4xAA battery box. Similarly, component costs would be significantly reduced, especially when purchased in bulk.

With the alterations of the design to make it more suitable for mass production now discussed, I can consider how it would actually be manufactured. If my product were to be produced commercially, I would expect numerically controlled drills to be used for the drilling of holes/vias in PCBs prior to the lamination stage where they are bonded together to form the multilayer PCBs. Pick and place robots would be used to populate the boards. These processes are beneficial when compared with the manual production methods I used for the prototype because they are automated processes and therefore carried out quicker and more accurately than human workers could, as well as being more reliable and not requiring pay. I would expect either reflow or wave soldering to be used for populating the boards. All processing components as well as the LEDs would be surface mount devices, while I/O such as the power supply, battery, menu/pit switches and LCD, being separate from the board, would be connected by through hole joints.

From an environmental and business point of view, it makes sense to use copper islands on the artwork. This is because they reduce the amount of copper to be etched, thus reducing etching chemical bills and increasing profits as well as reducing harmful wastes in the form of strong acids and alkalis. The use of CNC/robots is a double edged sword; while it is beneficial in that it reduces materials wastage due to failed products, it does also lower the amount of jobs created by the manufacturing operation.

With regard to quality of the final product, I would build quality assurance into the production of the games, by continuing with the techniques I used for the prototype – weaving holes for all flying components and heat-shrink on all bare joints. I would also conduct quality control. This would mean that all PCBs would be tested before being mounted in the cases and random samples of the completed products would be tested both for electrical functionality and defects in the casing (e.g. cracks/scratches/warping/discolouration etc.).

Industrial Practices, Social Issues, Systems & Control

As already stated, the case design would be altered to be a compact inject moulded construction in two parts - which would be assembled with the PCB and periphery components by hand.

I used CAD throughout the project – examples of CAD are Livewire used in the creation of schematics, PCB Wizard for the creation of art-works, MPLAB IDE for the development, testing and debugging of programs written in PIC assembly and C and ProDESKTOP for the design of the case. The advantages of CAD are that designs can be easily handled - for example, transferred by e-mail. Parts of the design stage can be automated, for example by the use of pattern commands for populating virtual PCBs with LEDs, and in the event that the design is updated or a new product revision is created, this can be accommodated by the CAMM manufacturing processes without changing equipment.

My product is a prototype, and with further development could be produced on a commercial scale.

Bibliography & Conclusion

Resources used

- 'PIC in Practice' (D.W. Smith 2002) - This introduced me to the general approaches used in writing programs with assembly code.
- www.cprogramming.com - This served as a reference for some C specific features such as pointers, as well as providing more information on function and variable declarations, as well as arrays.
- Wikipedia - This I used for information on industrial PCB production for the previous section.
- Microchip user forums - while I never created an account there, I was able to search the forums and find solutions for one or two of the more common problems that I had.
- Microchip online technical support centre - I created three support requests here, all of which are described in the relevant sections of this documentation.
- Microchip online datasheets - See appendix C.

Conclusion

On the whole I have really enjoyed the project, in particular the programming stages. It has been a great learning experience - prior to undertaking it I had never programmed in PIC assembly or C.

Appendix A (Rough Notes)

Throughout the project I have endeavoured to keep all rough notes together. They are included in this appendix.

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

APPENDIX A CONTENT ROUGH NOTES

Appendix B

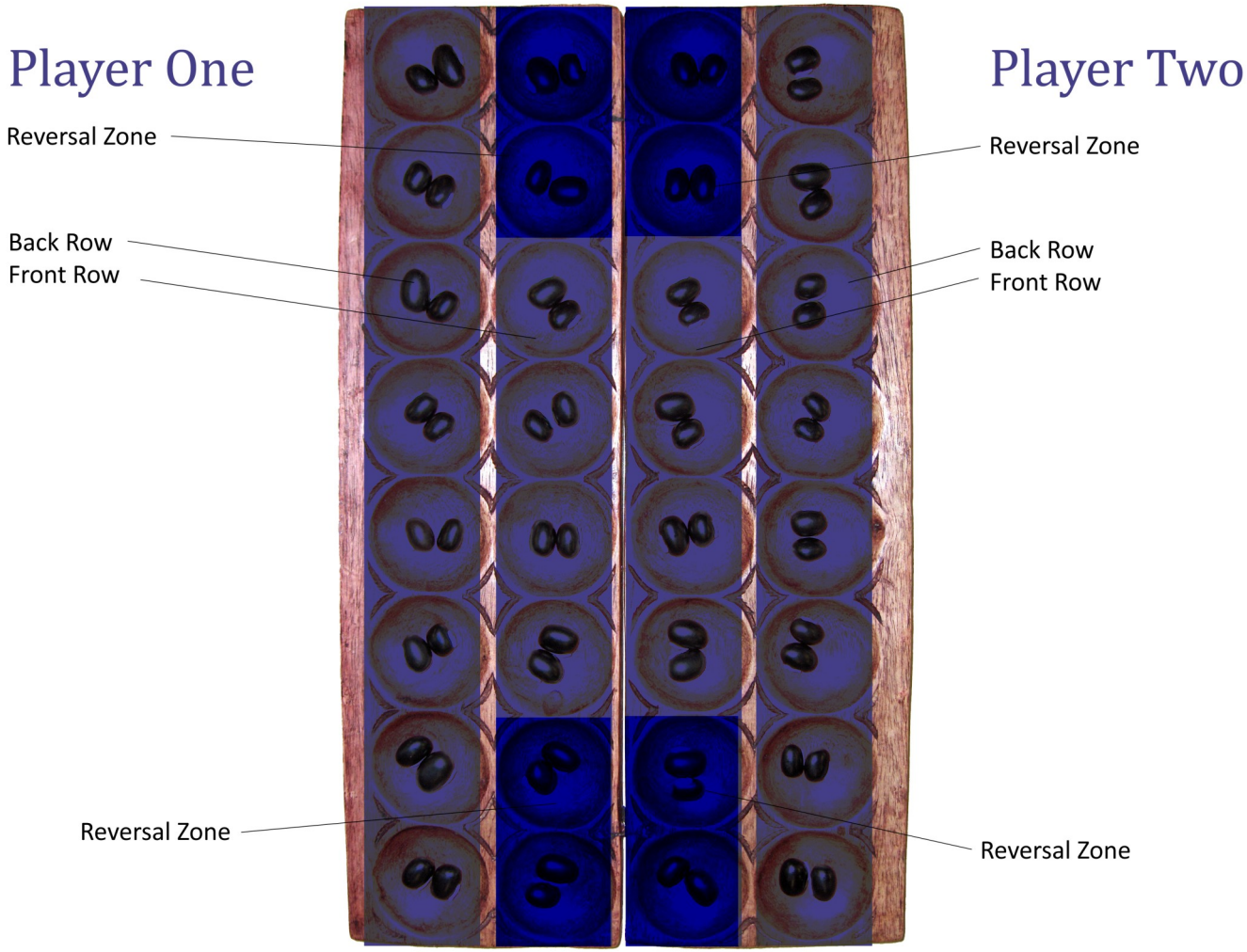
This appendix contains the following work done on my initial project idea, an electronic Bao board:

- Board Diagram
- Objective, Rules & Terms
- Preliminary Ideas & Analysis

In addition it contains work that I did towards the LED Register development that later proved useless (as the frequency of transmission I am using for the LED register control protocol is too high for the pulses to be distinguishable via standard PC sound equipment because the sampling rate is limited to 44,100Hz which is a time period of 22.6µs - when the smallest time period that would need to measured in my control protocol is 25µs - too close to the limit of resolution for it to have been of use:

- Use of laptop as an oscilloscope via the soundcard and the Audacity audio capture and editing program

Appendix B



Bao (Board Diagram)

Appendix B

Terms

Pits - The hollows in the board in which seeds are placed (there are sixteen per player in Bao).
Seeds - The pieces that the game are played with. In Bao, the game starts with two seeds per pit.
Rows - A row of pits. Each player has a front and back row. There are different rules for the front and back rows.

Objective

The aim of the game is to either empty the opponents front row or deprive them of any legal moves. This is achieved in accordance with the rules outlined below.

Rules

A players turn may consist of one or more moves. The following rules describe how to move the seeds:

1. At the start of a turn the player chooses a pit and direction.
2. The seeds from the pit are then taken into the hand and sowed around the board in the direction chosen, placing one seed in each pit passed until there are no seeds left in the hand. This sowing follows a loop that passes through the active player's front and back row. Seeds are never sown into the opponents pits.
3. If the last seed of a hand lands in a front row pit with one or more seeds already in, the contents (if any) of the corresponding pit in the opponents front row are taken, and sown starting from the end of the front row in the direction that play started in for that turn. If taking was a result of ending with the last seed of the hand in a reversal zone, the direction of play is first of all reversed.
4. If the last seed of a hand lands in any pit with one or more seeds already in it, the contents of that pit are then taken into the hand and another move follows, governed by these same rules. The direction of play is the same as that for the first move.
5. If the last seed of a hand lands in an empty pit, this is the end of the players turn.
6. Players can only take opponent's seeds in subsequent moves if they did so in the first move of the current turn.
7. The contents of a pit can only be played if there are two or more seeds in it.

Appendix B

The idea of the electronic version is to make the game quicker to play once the player has decided the pit and direction to start their turn is. This is possible because these are the only two variables that determine the outcome of a turn, and in Bao, turns can take literally hours/days given the correct starting conditions, as the sowing process just keeps on going. Therefore I decided that the functioning of the game would be as follows:

- Each pit has two backlit push button switches, one initiating clockwise play and the other anticlockwise play. The backlit is lit if is legal to move the seeds in that pit; if not legal, it is not lit. This helps the player more quickly analyse their options.
- Once the button has been pressed, the board will execute their move for them, though not instantaneously; there would be pauses between sowing each seed to allow the players to watch the move take place.

There are three main aspects of the system that need careful consideration:

- Input (telling it what you want to move and where)
- Output (showing how many seeds in each store)
- Processing (applying the rules of the game to the instructions given by the player, and moving the 'pieces' appropriately)

Input

As already stated, I planned to have two buttons per pit; this is a total of 4 x 8 x 2 = 64 switches (excluding auxiliaries, e.g. a 'New game' button). With so many switches, and given that simultaneous depressions are not necessary, the most obvious way of achieving connection of these to the microcontroller that applies the rules of the game is by some form of array scanning; with 64 switches, this could be achieved with 8 rows and 8 columns, which would require exactly 2 ports (one port = 8 bits) on a PICmicro.

Output

As the board consists of 32 pits, and each starts with 2 seeds, there are a total of 64 seeds on the board. Although all the seeds will never be in one single pit (the game would end before this), the maximum number that can be in a single pit is going to be close to 64 and therefore in solving the output problem, I have allowed for 64 seeds per pit.

Appendix B

There are a variety of output methods I could use for this, such as:

- 7-Seg displays
- LCD display
- LEDs

The problem with both 7-Seg displays and LCDs is that because they display textual representation of a number, they must be read the correct way up, and therefore I would require four 7-Seg displays per pit (two for tens and units, but doubled because I would need a pair pointing at each player). I haven't considered LCDs any further because, as with 7-Seg displays, they take away the quality of representation provided by LEDs; an array of LEDs, one per seed, gives a very quick graphical representation of the number of seeds in the pit.

Allowing for 64 seeds per pit and 32 pits, this is 32 x 64 = 2048 LEDs. Slightly concerned that this would be prohibitively expensive and time consuming to implement, I sought another solution; binary. Rather than having one LED for every seed that could be in a pit, I instead decided to represent the number in binary.

Since $\log_2 64 = 6$, a range of 64 seeds could be represented with only six LEDs per pit. That is up to 63 seeds (allowing for the display of zero seeds). Six LEDs per pit x 32 pits = 192 LEDs, a much more manageable number.

Unfortunately, doing it this way loses the quality of representation provided by using one LED per seed, though it does at least avoid the problem of opposing players having to read upside down (as there is no reading to be done). It also introduces another problem; players would need to be able to read, at a glance, a number from binary into decimal.

I decided that I could consider this a feature/twist of the electronic version, and continued developing my ideas. It was when I had just about finished these ideas, and was ready to begin formal documentation that I suddenly remembered the Bantumi mancala variant I mentioned in the project outline.

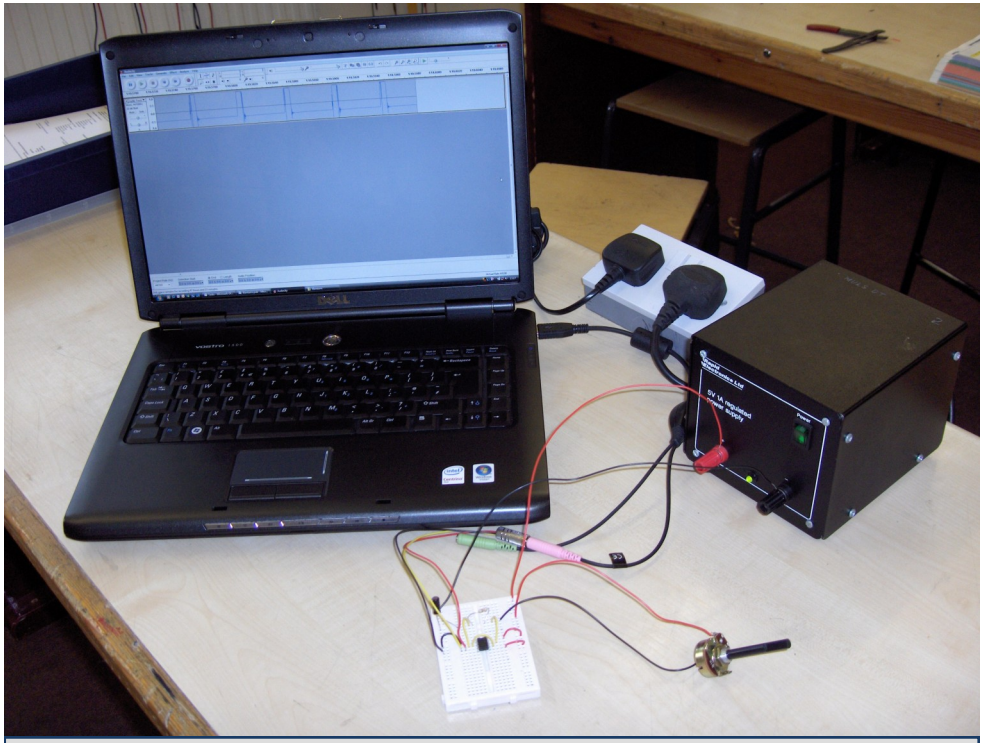
Though I had developed my ideas for this Bao variant to a fairly high level of detail, I have not formally documented them here because I no longer intend to produce such a game. For the purposes of completeness though I included them in this appendix.

Appendix B

While waiting for my order for a PICkit 2 programmer to come through, I started thinking about how I would go about developing the LED registers & control protocol.

Because the whole thing centres around a serial pulse train, it will be very important during development that I am able to easily see the waveform that is outputted by the controlling PICmicro. In order to this, I will need some form of oscilloscope. However, rather than borrowing a bulky CRO from the Physics department, I had the idea of using my the sound card on my laptop to do this, using an open source audio editing program called 'Audacity'. This will offer several advantages over a CRO:

- Will allow saving of traces, and post-examination
- Will allow reproduction of recorded traces (by clicking play!)
- Will allow construction of traces in software, and output via soundcard (rather than implementing a program in assembly code to do this)
- Since it's the same laptop I'm using for this documentation, I will not require any extra equipment
- Will allow me to insert traces directly into this documentation via print screen, rather than having to take a picture of a CRO screen!



Audacity receiving input from a 555 timer astable

With the benefits quite clear, I decided to start preparing and testing this technique ready for when the programmer arrives and I start programming. In order to interface a circuit on breadboard with the line-in of my laptop, I needed to know what the nominal line-level voltage is. I found this on the Wikipedia pages (http://en.wikipedia.org/wiki/Line_level and http://en.wikipedia.org/Nominal_level). In order to test it, I of course needed a sample input.

Appendix B

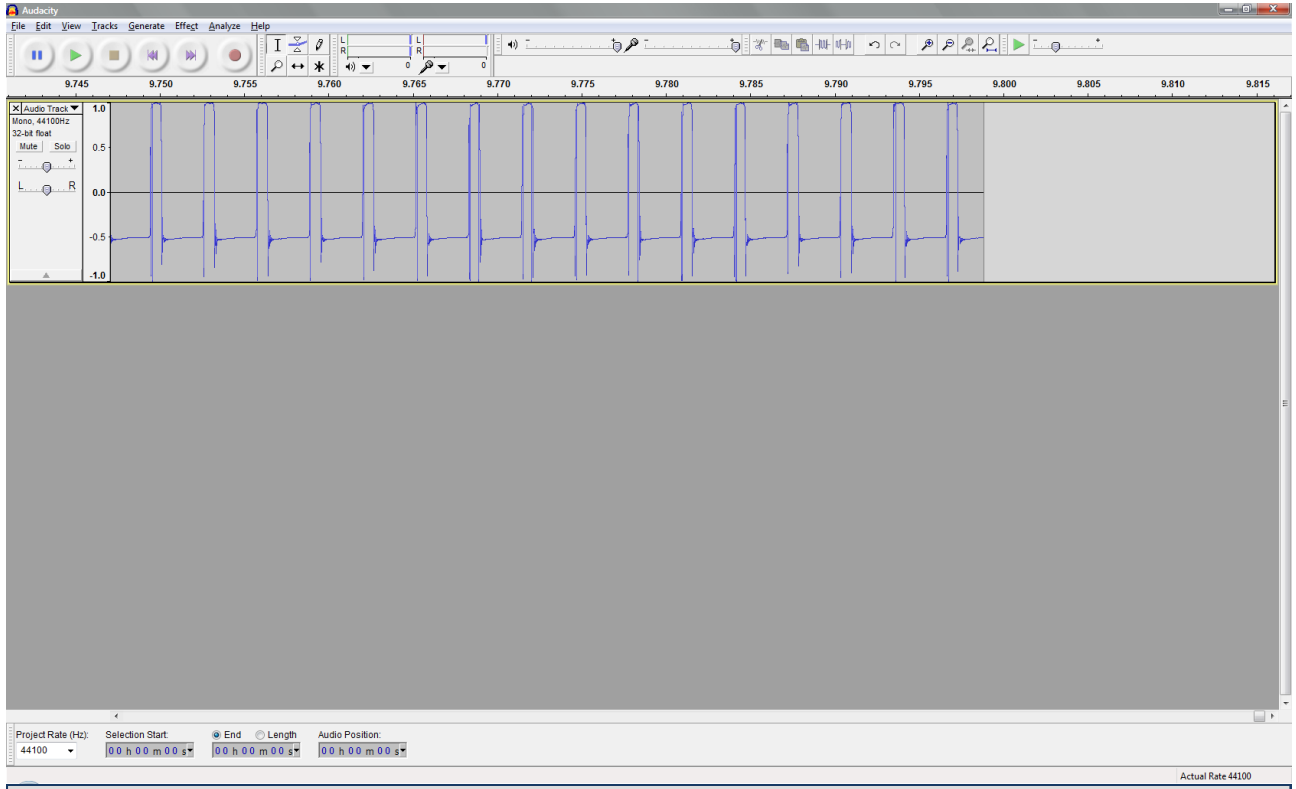
The input I chose to use as a means of testing this system was a 555 timer astable, using two 1kΩ resistors and a 1µF capacitor. Using the formula:

$$f = \frac{1.44}{C(R_1 + 2R_2)} = \frac{1.44}{1 \times 10^{-6}(1 \times 10^3 + 2 \times 10^3)} = 480Hz$$

$$T = \frac{1}{f} = \frac{1}{480} = 0.002083333s$$

This value matches with the difference on the time axis between two wavefronts on the Audacity waveform; it is only important that the timing is accurate for what I need.

I am therefore happy with this setup and conclude that it will be a very useful tool when I am developing the LED register protocol.



Audacity receiving input from a 555 timer astable

Appendix C

This appendix contains all datasheets that I read and use during the project. All but one (the PICAXE serial LCD module) are from Microchip, for Microchip products. The datasheets are listed in the following table:

Datasheet	*Pages*
C18 Getting Started	128
C18 Libraries	184
C18 User Guide	136
LCD Serial Driver	17
MPLAB Getting Started	44
MPLAB Quick Chart	8
MPLAB User Guide	288
PIC16F882 Datasheet	288
PIC18F252 Datasheet	332
PICkit 2 User Guide	58
Total:	1483

As can be seen there is a significant volume. Because I read so many different areas of them, it would be a unfeasible for me to select only those parts which I used for inclusion, and even if I did, I estimate this would still be several hundred pages worth. For this reason, I am only including two pages here - the pin diagrams for the PIC16F882 and PIC18F252. There are many other pages that were of great importance to me, but these two are those that I expect will of most use to the reader of this documentation.

For the sake of completeness, I have however included each datasheet on a CD which is included on the next page, along with all other files associated with the project (artworks, schematics, case design, program source code etc.).

Pin Diagrams – PIC16F882/883/886, 28-Pin PDIP, SOIC, SSOP

28-pin PDIP, SOIC, SSOP

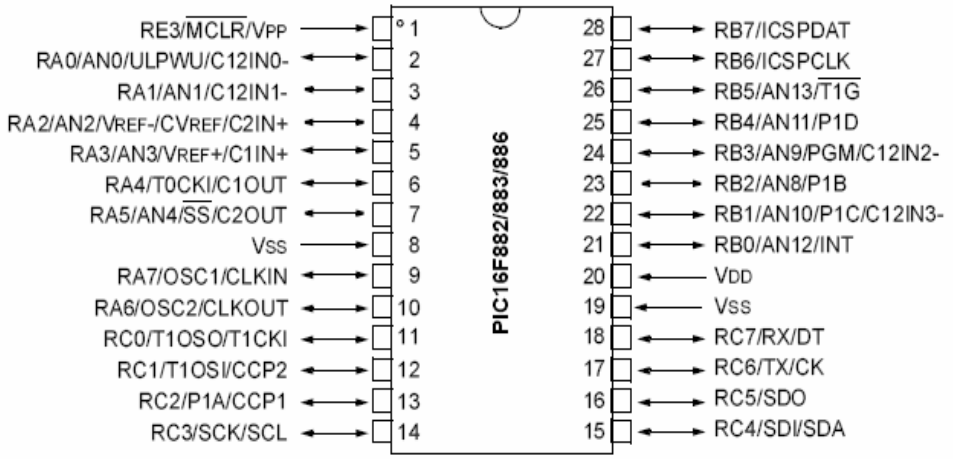


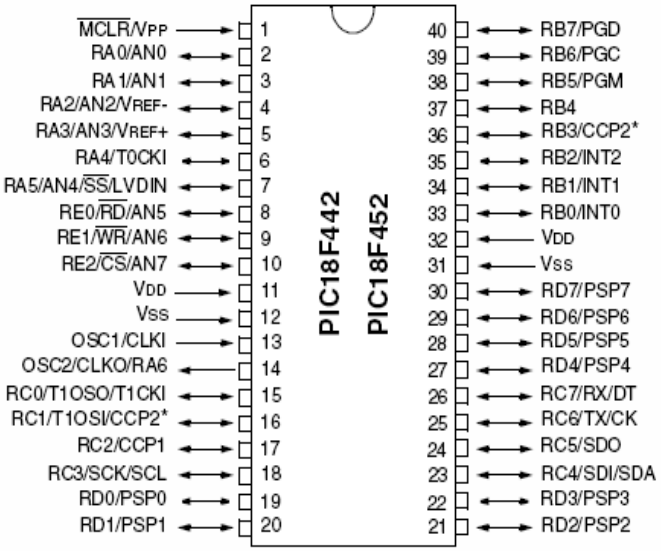
TABLE 1: PIC16F882/883/886 28-PIN SUMMARY (PDIP, SOIC, SSOP)

I/O	Pin	Analog	Comparators	Timers	ECCP	EUSART	MSSP	Interrupt	Pull-up	Basic
RA0	2	AN0/ULPWU	C12IN0-	—	—	—	—	—	—	—
RA1	3	AN1	C12IN1-	—	—	—	—	—	—	—
RA2	4	AN2	C2IN+	—	—	—	—	—	—	VREF-/CVREF
RA3	5	AN3	C1IN+	—	—	—	—	—	—	VREF+
RA4	6	—	C1OUT	T0CKI	—	—	—	—	—	—
RA5	7	AN4	C2OUT	—	—	—	SS	—	—	—
RA6	10	—	—	—	—	—	—	—	—	OSC2/CLKOUT
RA7	9	—	—	—	—	—	—	—	—	OSC1/CLKIN
RB0	21	AN12	—	—	—	—	—	IOC/INT	Y	—
RB1	22	AN10	C12IN3-	—	P1C	—	—	IOC	Y	—
RB2	23	AN8	—	—	P1B	—	—	IOC	Y	—
RB3	24	AN9	C12IN2-	—	—	—	—	IOC	Y	PGM
RB4	25	AN11	—	—	P1D	—	—	IOC	Y	—
RB5	26	AN13	—	T1G	—	—	—	IOC	Y	—
RB6	27	—	—	—	—	—	—	IOC	Y	ICSPCLK
RB7	28	—	—	—	—	—	—	IOC	Y	ICSPDAT
RC0	11	—	—	T1OSO/T1CKI	—	—	—	—	—	—
RC1	12	—	—	T1OSI	CCP2	—	—	—	—	—
RC2	13	—	—	—	CCP1/P1A	—	—	—	—	—
RC3	14	—	—	—	—	—	SCK/SCL	—	—	—
RC4	15	—	—	—	—	—	SDI/SDA	—	—	—
RC5	16	—	—	—	—	—	SDO	—	—	—
RC6	17	—	—	—	—	TX/CK	—	—	—	—
RC7	18	—	—	—	—	RX/DT	—	—	—	—
RE3	1	—	—	—	—	—	—	—	Y(1)	MCLR/VPP
—	20	—	—	—	—	—	—	—	—	VDD
—	8	—	—	—	—	—	—	—	—	VSS
—	19	—	—	—	—	—	—	—	—	VSS

Note 1: Pull-up activated only with external MCLR configuration.

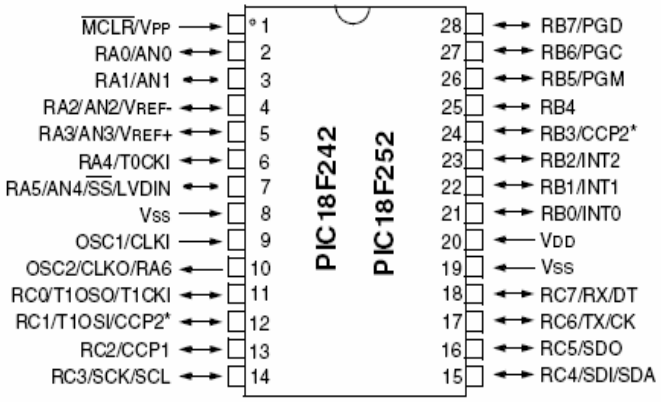
Pin Diagrams (Cont.'d)

DIP



Note: Pin compatible with 40-pin PIC16C7X devices.

DIP, SOIC



* RB3 is the alternate pin for the CCP2 pin multiplexing.

Appendix C