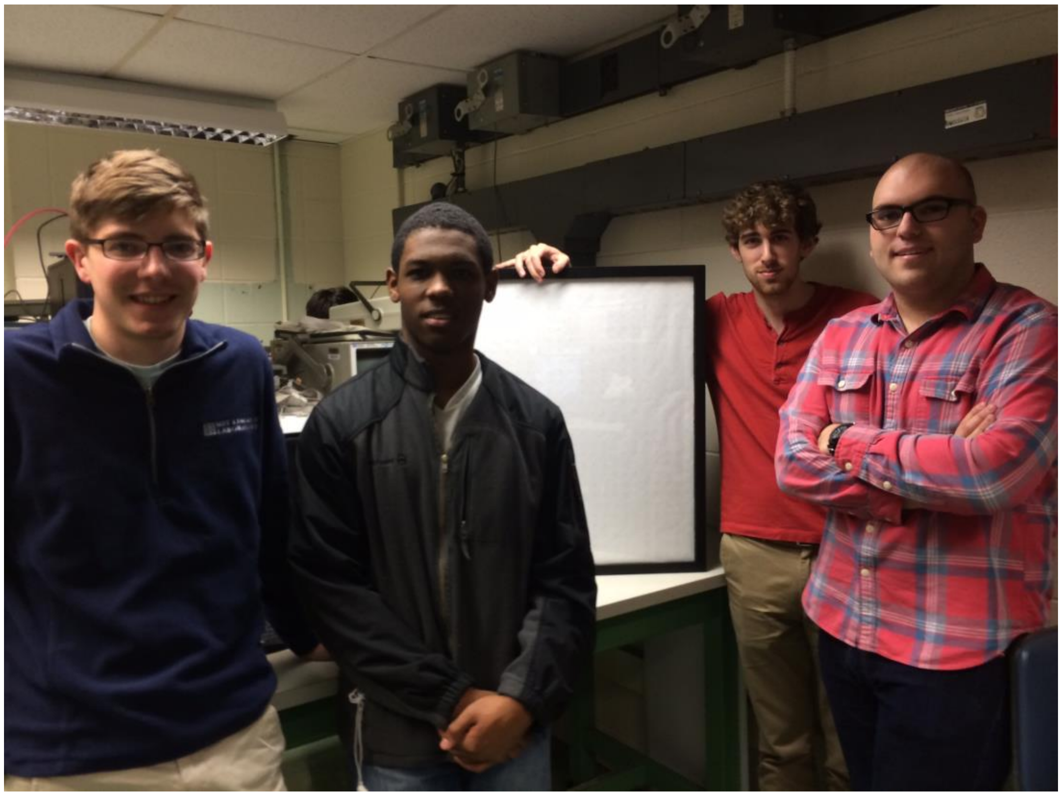
Life: Team 20



Team members, left to right are: Luke Walsh, Craig Kourtu, Jack Hammons, and Grant Gumina

Name	*Class No.*
Craig Kourtu	8711-K
Luke Walsh	0403-W
Grant Gumina	4371-G
Jack Hammons	2369-H

GRADING CRITERIA	MAX POINTS
Originality, creativity, level of project difficulty	20
Technical content, succinctness of report	10
Writing style, professionalism, references/citations	10
Demonstration of functionality	20
Overall quality/integration of finished product	10
Effective utilization of microcontroller resources	10
Significance of individual contributions*	20

scores assigned to individual team members may vary

Scoring Multiplier:

1.0	*Excellent – among the very best projects/reports completed this semester*
0.8 - 0.9	*Good – all requirements were amply satisfied*
0.6 - 0.7	*Average – some areas for improvement, but all basic requirements were satisfied*
0.4 - 0.5	*Below average – some basic requirements were not satisfied*
0.1 - 0.3	*Poor – very few of the project requirements were satisfied*

TABLE OF CONTENTS

1.0 Introduction

Life, is a wall mounted display of Conway's Game of Life. Conway's Game of Life, or Life as it's commonly called is a zero player game where once initial conditions are set no further input is required to make it run. Once a game state stops evolving (if initial conditions permit that), the game board programmatically resets itself. Interaction with the game is watching how the game board evolves over time. *Life* is intended to be an indoor piece of art and therefore much effort was put into building as polished of a product as possible.

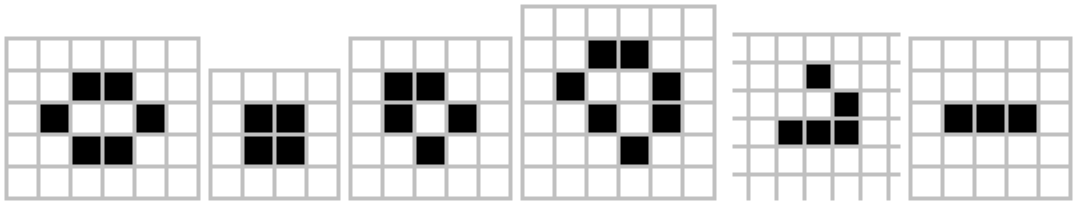
Our display consists of a rectangular 21" x 27" LED matrix, the game board, residing in a 22" x 28" shadowbox where each LED represents a cell. A lit LED represents a live cell and a LED turned off represents a dead cell. Each cell follows a specific set of rules which determines whether or not it will be alive (on) or dead (off):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbor's lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Reference [5] explains Conway's Game of Life, and the mathematical field of study behind it, in much more detail.

The initial pattern, or seed, that starts the game is generated by assigning each cell the value read by the onboard Analog-To-Digital (ATD) converter on a floating port on the Freescale MC9S12C32. That value is them is modded by 2 so a 1 (alive/on) or 0 (dead/off) value is assigned to each cell. This results in a pseudorandom game board generated each time the game is initialized or reset.

Since the game self generates based on initial conditions, over time various patterns form and some even persist indefinitely. Some examples are below:



(Figure names from left to right: beehive, block, boat, loaf, glider, blinker)

Some of the figures seen above will persist indefinitely (beehive, block, boat, loaf) while others can take on multiple forms in their evolutionary process (glider, blinker).

Given the scale of this project, work was divided up among the four group members as follows:

Hardware – Jack Hammons & Craig Kourtu

- LED matrix construction – Jack Hammons
- Packaging construction and assembly – Jack Hammons & Craig Kourtu

- LED matrix and PCB debugging – Jack Hammons & Craig Kourtu

PCB – Luke Walsh (Honors Contract)
Software - Grant Gumina & Luke Walsh

- Implementation of Conway's Game of Life algorithm – Grant Gumina
- SPI abstraction layer – Grant Gumina
- SPI communications – Luke Walsh & Grant Gumina
- ATD pseudo random board generation – Luke Walsh

Report – Craig Kourtu, Grant Gumina, Jack Hammons, and Luke Walsh

2.0 Interface Design

Our goal was to create a project so simple even our friends that Liberals Arts majors and healthcare.gov contractors could use it. To meet these requirements, the only physical interaction users have with *Life* is plugging it in. Upon being plugged into a wall socket, *Life* immediately starts evolving from the random seed state created on startup of the microcontroller.

The initial state is set on the LED matrix which gets updated as evolution takes place on the game board. Values are set on the matrix with two 32-bit shift registers. Datasheet [4] explains how the PLDs can be interfaced together to form one larger shift register. Datasheet [1] explains how data is able to be sent to the shift registers via SPI. One register sets the column values of the matrix and the other sets the values of the rows.

A chip select bit is used to determine which shift register is being sent data. Bringing a cell to life (turning on an LED) requires the value of the column to be 0 and the value of the row to be 1. This way current is allowed to flow and the LED can turn on. The Serial Port Interface (SPI) abstraction layer written (and described later on in this report) allows for any LED to be turned on just by supplying the proper coordinates. The same functionality can be done for turning off an LED.

To keep the game interesting, every time the board resets a new randomly generated state (alive/dead) must be assigned to each cell. In order to achieve this, the Analog-To-Digital converter was used in *Life* when determining an initial game state. For each cell in the game board, the ATD reads in a floating port and assigns the value pulled to a variable which is then modded by 2. The value of the ATD % 2 is assigned to the cell being iterated over. This way a 1 or 0 gets "randomly" assigned to the cell.

Much of the software logic behind the ATD functions was built around the user guide [2] for the microcontroller.

Lastly, it is important to note the conscientious decision of this group to avoid allowing the user to interact with *Life*. Since *Life* was designed not only as a 362 project, but also as a beautiful piece of art. It was believed that including the ability for users to interact with *Life* would pervert the purity and beauty of the project as *Life* evolves is all about not being in control. The game board randomly generates and continues to evolve at a set pace until it no longer can. There is a certain beauty in that which the group strived to maintain and protect throughout the project.

3.0 Microcontroller Resource Utilization

The SPI, or the serial port interface, allows us to update our shift registers by pushing values from the updated matrix a given rate. Each initialization is written below and rationales are provided in the form of comments.

SPICR1 (SPI control register 1)
SPIE = 0 // No need for SPI interrupts.
SPE = 1 // SPI system enabled turns on up the SPI.
SPTEF = 0 // No need for SPTEF interrupts.
MSTR = 1 // The SPI is in master mode, because it is pushing values out to the shift registers.
CPOL = 0 // Active high clock, default value and no reason to change it.
CPHA = 1 // Data sampled occurs at even edges.
SSOE = 0 // Slave Select output disabled, not operating in slave mode thereby unnecessary (We manually select the shift registers to push out
LSBFE = 1 // We push out least significant bits first so the junk bits that aren't needed for our display fall off the shift register leaving only the bits we need.

SPICR2 (SPI control register 2)
MODFEN = 0 // Slave select, port pin unnecessary due to only one slave
BIDIROE = 0 // Output buffer
SPC0 = 0 // Bi-directional mode not needed
The ATD will be used to change the rate the screen refreshes or the rate of evolution.

RTI (Real Time Interrupt) and TIM (timer)
```
RTICTL = 0x1F;     // setting up the RTI for a 1ms interrupt rate
CRGINT = 0x80;
TSCR1 = 0x80;      // enable TC7
TSCR2 = 0x0c;      // set TIM prescale factor to 16 and enable counter reset after OC7
TIOS = 0x80;       // set TIM TC7 for Output Compare
TIE = 0x80;        // enable TIM TC7 interrupt
TC7 = 1;           // set up TIM TC7 to generate {batshit insane} interrupt rate
```

ATDCTL2
ADPU = 1 // Turn on the ATD
AFFC = 0// we're not using fast flag CCF clearing mode
ASCIE = 0 // Disables the ATD interrupts
ATDCTL 3
Conversion Sequence length was set to one, allowing for the quickest conversion as possible. Since nothing is extremely time critical any value (1 through 8) would work.
ATDCTL4
All of the below are set to zero, because defaults we've used all semester.
DJM = 0 // Sets data to be left justified
DSGN = 0 // unsigned data
SCAN = 0 // single conversion (Never used scan mode, and not needed now)
MULT = 0 // Only Using one channel

Software Narrative

Implementing Conway's Game of Life on the MC9S12C32 microcontroller was quite challenging. It required several layers of abstraction in order to keep the fundamental game algorithms intact, as well as some slight algorithm optimizations in order to compensate for SPI write out times and propagation delays. Code functions are categorized below. The table is ordered from highest level functions to lowest level functions.

Function	Description	Function Category
turnOnRow(int)	Turn on a given row of LEDs	SPI abstraction
turnOnCol(int)	Turn on a given column of LEDs	SPI abstraction
turnOffLEDs(void)	Turn off all LEDs in the matrix	SPI abstraction
resetGame(void)	Clear game board and start over	Game board manipulation
evolve(void)	Evolve the game board	Game board manipulation
displayBoard(void)	Show game board values	SPI abstraction
displayColumn(int)	Show values of board for given column	SPI abstraction
setSPIDataZero(void)	Set SPI register all to 0s	SPI manipulation
setSPIDataOnes(void)	Set SPI register all to 1s	SPI manipulation
setSPIDataBit(int, int)	Set a specific bit of SPI register	SPI manipulation
shiftOutX(void)	Set chip select for X register and shiftout	SPI abstraction
shiftOutY(void)	Set chip select for Y register and shiftout	SPI abstraction
shiftOut(void)	Shift SPI register to the GAL arrays	SPI manipulation

As mentioned elsewhere, the SPI functions were abstracted to the point where they could be dropped directly into the Conway's Game of Life algorithm. The evolve function was called only when a "evolve flag" was set. This flag was set inside the RTI and called every second resulting in the user being presented with an updated game board every second.

The evolve function runs through the game's grid, up to down, left to right using two for loops. While on a given index of the grid, two more four loops check the squares around it to determine the square next state. To do this we must have two double matrices to represent the current and next states of the matrix. The evolve function then checks the current state and writes to the next state which is just a temporary variable.

Importantly, each time the evolve function runs the program checks if there haven't been any changes made to the game board since the last evolution. If that's the case, then the simulation has run to its logical end in which case the program restarts the simulation. Life goes on.

After each evolution, the game board is displayed via the displayBoard() function. This function is an abstraction of many SPI functions which are abstractions of even more lower level SPI code. Basically, displayBoard() looks at the game board array of arrays and iterates over every column in the LED matrx lighting up the specific rows (LEDs) needed. It does this fast enough that humans cannot see the LEDs turning off after displayBoard() iterates over a column.

4.0 Packaging Design

The packaging is a black shadow box that holds the LED matrix and PCB responsible for displaying the game board. The matrix is made of 567 white LEDs, with 270 ohm resistors attached to them. The shadow box's dimensions are 22" x 28" x 4". Each column and row of LED's are connected together by a wires each leading to a 32-bit shift register built from 3 GAL22V10Ds. Columns go to the shift register denoted as X, rows go to the shift register denoted as Y.

Making up the LED matrix display are the LEDs mounted inside two sheets of 1" peg board and hot glued in place. Importantly, each row of LEDs/resistors was insulated by a coating of hot glue. This way the columns can be set on top of them without having to worry about shorting any cell. Over 65 glue sticks were used in the construction of *Life.*

A 21.5" x 27.5" sheet of diffusion paper was placed in between the LED matrix display and glass pane of the shadow box. The diffusion paper allows users to look at the display directly without being blinded since the cell lighting appears to be much softer.

Besides the LED matrix display, the PCB was placed inside the shadow box. The PCB dimensions are 8" x 8" with traces on both sides of the board. For each surface mounted component, besides the decoupling capacitors for each shift register, a surface mounted socket clamp was soldered to the board. This allowed the team to easily take components out to test if they were damaged or programmed incorrectly. Each column and row of the LED matrix has a wire attached to it leading from the matrix assembly to a port in the PCB. Columns are assigned to one port and rows to another. For rows, only ports 1 – 21 are utilized. For columns, only ports 1 – 27 are utilized. Each shift register has extra ports. As mentioned in another section, the team made the design decision to include extra ports to accommodate a larger display case.

The PCB was mounted on a 20" x 26" piece of insulating foam board to prevent short-circuiting and damage. This foam board was then glued inside the display casing with supporting rods in between the foam and the peg board. This allowed for the PCB to be mounted vertically. The power cable was also soldered directly onto the board instead of using the adapter shown later on in the documentation. That was only used for testing the circuit.

Playing into the team's decision to make *Life* beautify and as clean as possible, a single black power cable comes out from the back of the casing. Users can utilize the wall mount holes in the casing to mount *Life* anywhere it can be supported. The project weights ~10 lbs, so it's only recommended to hang it on something the user knows for sure can support that kind of weight.

5.0 Summary and Conclusions

From the beginning of the project, we knew that *Life* was a tremendous undertaking due to the sheer scale of the project. Consequently, we spent a lot of time planning and designing the project. The physical construction of the display was paramount, which is why we spent a lot of time and money exploring all possibilities of what materials we could use/implement. This included driving to Indianapolis in search of lighting louvers, spending a weekend at the Home Depot looking for frame materials, and spending 35 hours soldering together the LED matrix on which success of the project hinged.

The PCB was an undertaking in and of itself since only one group member had previous experiencing designing PCBs. Luke pulled two “all nighters” in designing the PCB, with much of the time spent “fighting” the Eagle design software. Importantly, the design decision was made to include up to 32 ports for each of the two shift registers knowing that it was highly unlikely they would all be utilized. This decision was made since at the time, because the team hadn’t decided on *Life’s* casing.

Using a PCB ended up saving us a lot of time since we didn’t have to build/test a circuit on a messy and cluttered breadboard. Many other groups had to deal with fitting their entire circuit on a single breadboard or otherwise wiring together several breadboards. It was observed that these groups spent a lot of time debugging shorted or broken connections each time they moved their project. Using a PCB eliminated all of those problems. In the end, we believe using a PCB saved the team a lot of time.

We also utilized the source code control system Git to store our code. After each major milestone or bug fix, we would push to the repository so progress would not be lost. This ability to quickly push and revert changes proved to be a major advantage as coding sessions would go into the early hours of the morning.

The biggest take away from this project was how important planning and time management were to the success of an undertaking of this scale. We were glad to have started designing the project before break and writing as much software as possible during Thanksgiving break.

If we had more time and resources, we would have increased the brightness of the LED matrix by removing the 270 ohm resistors in series with each LED. We would have also increased the power supply to 7 volts using a transformer.

We also would have investigated a better form of diffusion paper that would allow for more light to be seen. Also, we would have spent more time mounting the LED matrix inside the shadow box. Currently, it is attached by hot glue mounts which we’re worried will deteriorate over time.

Lastly, we would have slightly altered our PCB design by reducing the board size and swapping out the GAL22V10D chips for an actual shift register IC. This would have reduced cost and sped up development time by eliminating the need to write ABLE code.

6.0 References

[1] S12CPUV2 - Microcontroller - Datasheet from Freescale
https://engineering.purdue.edu/ece362/Refs/9S12C_Refs/S12CPUV2.pdf

[2] MC9S12GC – Microcontroller – User Guide from Freescale
https://engineering.purdue.edu/ece362/Refs/9S12C_Refs/9S12C128DGV1.pdf

[3] GAL22V10D – Programmable Logic Device – User Guide from Advanced Micro Devices
https://engineering.purdue.edu/ece362/Refs/Pld/pal_basics.pdf\

[4] GAL22V10D - Programmable Logic Device – Datasheet from Advanced Micro Devices
https://engineering.purdue.edu/ece362/Refs/Pld/palce22v10.pdf

[5] Gardner, M. (1978). *Aha! Insight*. New York: Scientific American/Freeman.

Appendix A:

Individual Contributions and Activity Logs

Activity Log for: Grant J. Gumina **Role:** Software & Documentation

Activity	*Date*	*Start Time*	*End Time*	*Time Spent*
Started writing game logic	11/22/2013	8pm	12pm	4hrs
Finished writing game logic	11/23/2013	1am	11am	10hrs
Modified louver	11/23/2013	11am	5pm	6hrs
Wrote game board initialization	12/2/2013	8am	12pm	4hrs
Built/Debugged hardware & PCB	12/2/2013	1pm	1am	8hrs
Wrote SPI abstraction functions and debugged lower level communications issues	12/3/2013	10am	10pm	8hrs
Wrote final report and documentation (coding comments etc.)	12/3/2013	11pm	4am	5hrs
Driving to Indianapolis to get a louver	11/27/2013	12pm	3pm	3hrs

Written Summary of Technical Contributions: Grant J. Gumina

Mainly, my contributions focused on the software, construction of physical packaging, and writing the documentation. It was my job to port the Conway's Game of Life algorithm to the Freescale MC9S12C32 microcontroller as well as contribute to software architecture of the project.

Porting the algorithms for Conway's Game of Life was initially a challenge. First I wrote the algorithm in a Unix environment and made sure the logic was correct. This allowed me to speed up the debugging and development time tremendously when compared to debugging this same code on the microcontroller. Then I took this code and put it inside the microcontroller. After making some optimizations to the algorithm (for exampling changing the double integer array to a double char array), the code was able to run and Conway's Game of Life was able to be run on our microcontroller.

The abstraction functions I wrote made it very simple to drop in the SPI code to the Conway's Game of Life algorithm without much modification. This sped up the development process greatly as well as enabled the team to unit test code more easily. Specifically, by modularizing the functions that set the game board arrays, set the SPI registers, and actually transmitted the SPI data to the registers, we were able to easily and quickly debug issues.

For example, we were easily able to determine there was an issue with our LED matrix when a few LEDs would not light up by simply setting those LEDs to high from inside our code. Another example was when we were worried about the intensity of our LEDs. We were able to simply change the SPI shiftout delay as well as the delay rate of our evolve function separately. This allowed us to maximize the brightness of the display. The result was a much better viewing experience for the users.

When building the physical packaging, much of my time was spent on building an element we ended up not using. Initially we were very worried about light leaking between the LEDs since building a project of high quality was a major concern of ours. Specifically, we were going to use a lighting louver found in many buildings ceiling lighting as a grid for our LEDs. The idea was to put the louver between the peg board (where LEDs were to be placed in) and the diffusion paper/glass of the shadow box. After spending an entire day modifying the louver, a decision was made that the louver was not going to be able to be implemented. Instead we used two peg boards to prevent light leakage. The only downside was that our cells were not square – instead they were round.

Lastly, I contributed to the documentation and reporting efforts by building accurate flowcharts, commenting my code contributions, and detailing the technical details of the algorithms used. I was also able to contribute by frequently taking videos of the *Life's* progress as we built this project.

Activity Log for: Craig Kourtu **Role:** Hardware & Documentation

Activity	***Date***	***Start Time***	***End Time***	***Time Spent***
Brainstorming, purchasing materials, etc	11/21/ 2013	11:30 am	3:30 pm	4 hours
Modification of louver	11/22/ 2013	8:00 pm	11: 30 pm	3 hours thirty minutes
Built/Debugged hardware & PCB	12/2/2 013	1:00pm	1:00am	12 hours
Debugged low level SPI issues	12/3/2 013	9:00 am	10:00 pm	13 hours
Documentation	12/4/2 013	1:00 pm	3:00 pm	2 hourd

Written Summary of Technical Contributions: Craig Kourtu

The main focus of my contributions to the project was hooking up the PCB to the grid and making sure all of the shift registers were pushing values quickly enough to create an accurate display. I consulted Luke Walsh on the creation of the PCB, and assembled it with the shift registers that we were to use to push values to our display.

My biggest contribution was debugging an error we saw between shifting between the y shift registers and the x shift registers. The chip enable signal we were using was taking too long to propagate, leading the shift register rejecting any changes we pushed to it. To fix this a delay had to be written into the shift out functions to give them shift reactions time to “unlatch” its flip flops. I also assisted with the writing of the SPI abstraction.

Another major part of the project was the creating the ideal way of packaging it. I put forth the idea that a shadow box would be ideal mix of aesthetics and space, and assembled the final packaging (not the led grid) including mounting the board with a insulating back).

Initially we were worried about light leakage so I spent a significant amount of time modifying a lighting louver to create a grid to put of LED, as to isolate them from one another, in the end that wasn’t needed, as a peg board was easier and worked just as well.

Lastly, I wrote much of the documentation, giving detailed description of what our project is, how it was built and interfaced, how we use our peripherals, and a summary of our design.

Activity Log for: Jack Hammons **Role:** Hardware & Documentation

Activity	*Date*	*Start Time*	*End Time*	*Time Spent*
Debugging hardware	12/4/13	12pm	6pm	6hr
Debugging hardware	12/3/13	5pm	11pm	6hr
LED Matrix construction (soldering, gluing, twisting, etc.)	11/27/13 to 12/1/13	12pm	4am	30hr
Driving to Indy for parts	11/20/13	3pm	6pm	3hr
Research/Parts acquisition	11/17/13	10am	4pm	6hr

Written Summary of Technical Contributions: Jack Hammons

My part of the project centered around the design of the packaging, construction of circuitry, researching and acquiring parts, debugging the hardware and planning the logic of the code.

When designing the packaging I strove to create a housing that would give the project an aesthetic appeal. To do this I went to several different stores in search of frame to hold the LED grid. Once a frame was found I went to Home Depot and purchased peg board. The idea of the peg board was that the LEDs would be stuck through the holes and soldered together on the back. To give the grid a more defined look I decided to place a louver on the front. This would separate the contributions of the individual LEDs and make the pattern of light more apparent.

To construct the circuitry I used a method of twisting together the lines from the shift registers, resistors and LEDs and then covering these with hot glue. This methodology was found through a process of trial and error and although I was initially skeptical of its effectiveness it has proven to work very well. Hooking everything together was by far the most work intensive part of the project.

In order to find all of the odd parts associated with the project it was necessary to do a fair amount of research. The louver was found at a firm in Indianapolis so it was necessary to drive down and get it. Other parts were located in a more convenient manner. The frame was found at Hobby Lobby and many of the other parts were located at Home Depot.

Debugging the hardware proved another tricky part of the project. When everything was hooked up the led matrix would not light up. After several hours a solution was found. As it turns out the PCB board is designed in such a way that it is not possible to program the micro controller without first removing it from the board. Once this was discovered the project progressed merrily on its way.

The very first part of the project to which I contributed was the planning of the code architecture. To do this several flow charts were created with the express purpose of determining a way to to write the code. This proved very useful later on in the project when we needed a standard from which to base our program.

Activity Log for: Luke Walsh **Role:** PCB & Software

Activity	***Date***	***Start Time***	***End Time***	***Time Spent***
Initial PCB Design	11/11	6:00pm	3:00am	9hrs
PCB Review with Prof. Meyer	11/12	4:00pm	4:30pm	.5hrs
Second PCB Design	11/14	8:00pm	4:30am	8.5hrs
PCB Review with Prof. Meyer	11/15	4:00pm	4:30pm	.5hrs
PCB Fixes and submission	11/15	5:00pm	6:30pm	1.5hrs
Purchasing parts for the frame	11/16	10:00am	2:00pm	4hrs
Drive to Indy to purchase light diffuser	11/19	3:00pm	5:00pm	2hrs
Developing interface logic and debugging PCB	12/2	2:00pm	11:30pm	9.5hrs
Debugging interface logic and integrating with game logic	12/3	12:30pm	10:00pm	9.5hrs
Writing documentation	12/4	3:30pm	5:00pm	1.5hrs

Written Summary of Technical Contributions: Luke Walsh

For this project I was in charge of the hardware design. I opted to design a printed circuit board because the team felt that it would be useful to have all of the hardware consolidated on one board rather than having to wire everything by hand on the breadboard.

I decided to use sockets for the shift registers as well as the micro controller so that we would be able to reprogram them at any time in case our first take at the code was not correct. This proved to be invaluable as we constantly removed the micro controller to reprogram it for convenience once the PCB was mounted on the frame.

To control individual LEDs I designed a system in a grid, where each row or column could be set to high or low. This allowed us to select one column at a time to sink current, and turn on the correct rows for that column. By shifting which column was selected very quickly (>100Hz) it is not visible to the eye that there is really only one column on at a time. This is similar to how television displays work.

The actual frame was a set of LEDs and resistors in series that Jack soldered and hot glued to prevent accidental shorts. My PCB provided holes all around the outside where I soldered the wires for each column and row. I opted for the maximum possible size PCB to allow us to have ample room for attaching all of the wires. This ended up being useful because we were able to test the connectivity before soldering on the wires to test the shift logic.

I programmed 6 GAL22V10s, three for the rows and 3 for the columns. I tied all of them to a port pin on the microcontroller to act as a chip select. Data was shifted out using SPI, and would go to one set of cascaded shift registers based on the state of the chip select pin. Our hardware was capable at maximum of driving 900 LEDs, in a 30x30 grid. We ended up using a 27x21 grid because it fit nicely in to a portrait frame.

I decided to use generic PLDs instead of dedicated shift registers to allow us more flexibility if we decided that we wanted to change our hardware logic at all. We didn’t end up doing this, so the design could be copied by using dedicated shift registers rather than custom programmable logic devices.

I also wrote the interface logic for the hardware so that specific LEDs could be turned on. This included a timing analysis of the GAL22V10 since we had to allow time for the chip select changes to propagate to the devices. We used a game loop that would constantly update the display by shifting through each column and displaying the correct high and low state based on the state of the game.

By designing the code in this way I was able to test and debug the interface logic without needing to know how the game was implemented. This modular software design allowed our group to iterate quickly, and apply lean principles to our project management.

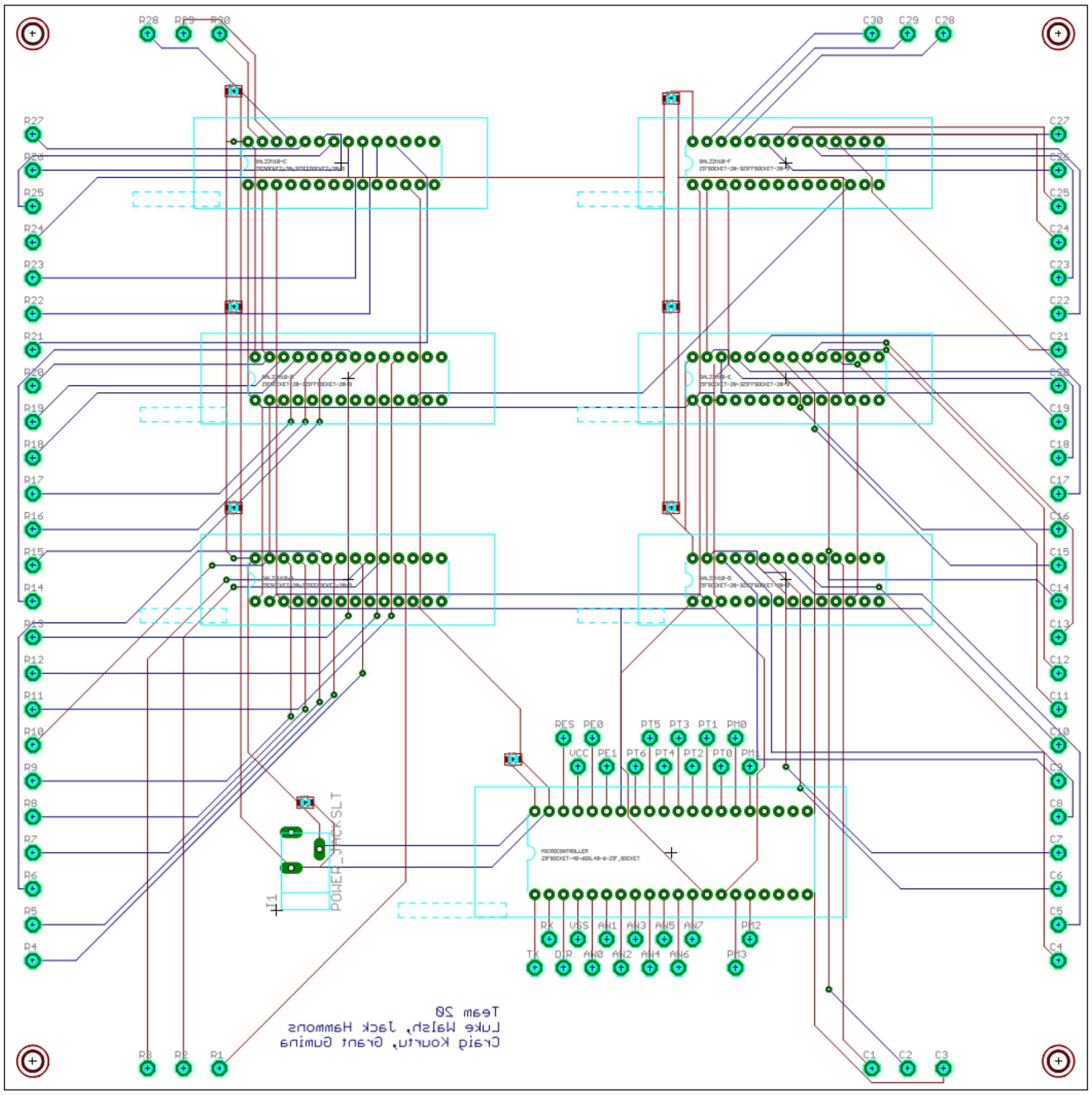
Once the interface logic was completed I passed this off to Grant who wrote the software to generate the constantly updating board to display the game of life.

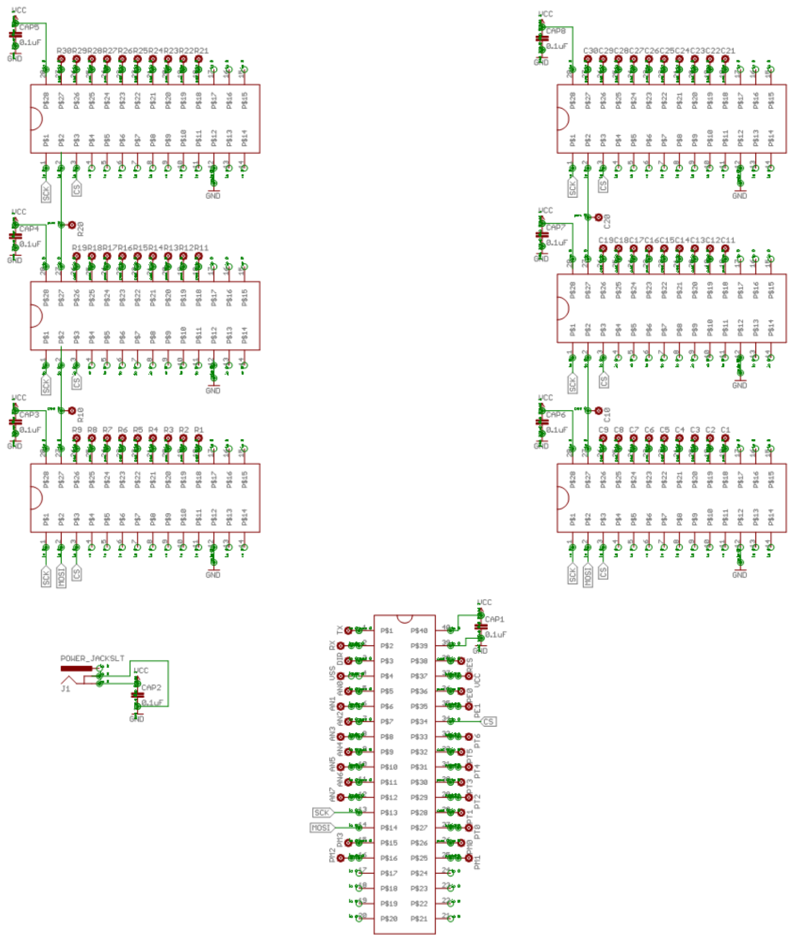
Appendix B:

Interface Schematic

Life OrCAD Schematic
Luke Walsh

The PCB schematic is shown below.


R28
R29
R30
C30
C29
C28
R27 R26 R25 R24 R23 R22 R21 R20 R19 R18 R17 R16 R15 R14 R13 R12 R11 R10 R9 R8 R7 R6 R5 R4
C27 C26 C25 C24 C23 C22 C21 C20 C19 C18 C17 C16 C15 C14 C13 C12 C11 C10 C9 C8 C7 C6 C5 C4
RES PE0 PT5 PT3 PT1 PM0
VCC PE1 PT6 PT4 PT2 PT0 PM1
POWER_JACKSLT
MICROCONTROLLER
RX VSS AN1 AN3 AN5 AN7 PM2
TX DIR AN0 AN2 AN4 AN6 PM3
Team 20
Luke Walsh, Jack Hammons
Craig Kourtu, Grant Gumina
R3 R2 R1
C1 C2 C3


VCC
CAP5
0.1uF
GND
CAP4
CAP3
CAP8
CAP7
CAP6
CAP1
CAP2
POWER_JACKSLT
J1
SCK
MOSI
CS
R20
R10
C20
C10
TX
RX
PM2
PM3
PT0
PT1
PT2
PT3
PT4
PT5
PT6
PM1
PE1
AN1
AN2
AN3
AN5
AN6
AN7

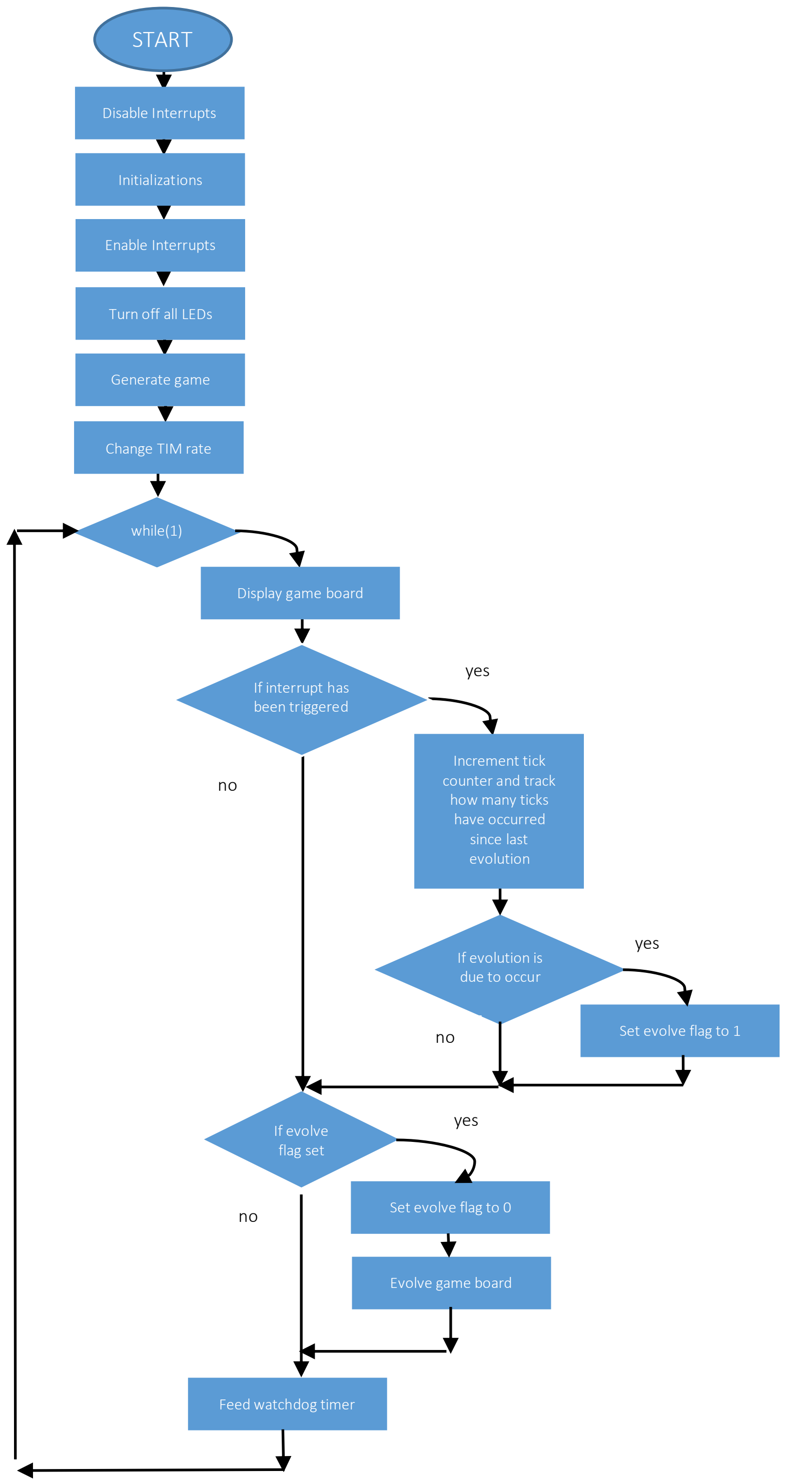
Appendix C:

Software Flowcharts

Life Source Code Flow Charts

Grant Gumina

main.c

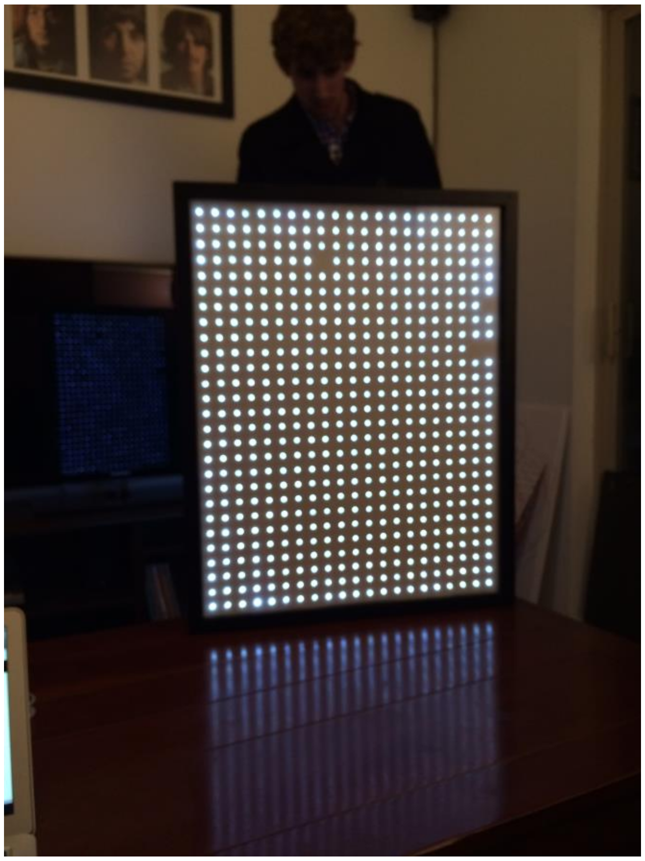


Appendix D:

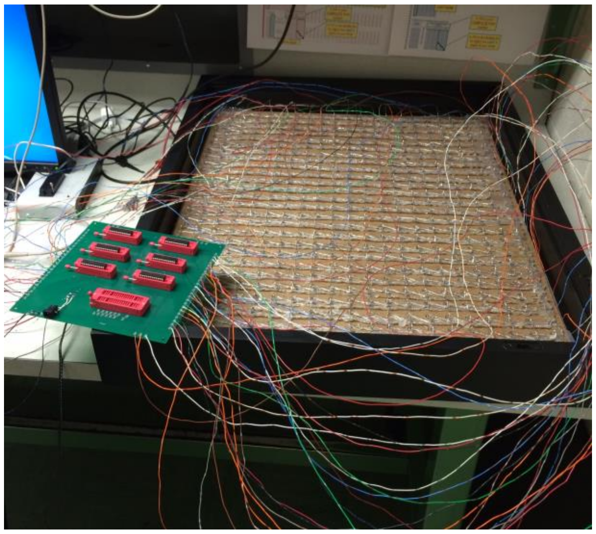
Packaging Design

Life Packaging and Fabrication Documentation
Grant Gumina

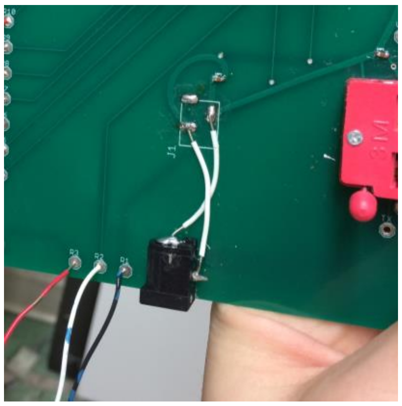
Initial tests were performed over Thanksgiving break to determine whether or not the LED matrix was functional. Only a few LEDs had bad connections which were easily resolved.



Almost 600 LEDs were used in *Life's* game board. They were controlled by two 32-bit shift registers, each composed of 3 GAL22V10D ICs. The PCB (with all 6 shift registers clamped in their sockets) and LED matrix can be seen below.



Our PCB design was partially incorrect as we accidentally flipped ground and power pads for our power jack. To overcome this, we soldered a new power jack connection to the PCB.



Rows and columns were assigned to different shift registers. Each row/column had a corresponding port on one of the 32-bit shift registers, as seen below.