University of Strathclyde Engineering

Mini-grid Game

Liam Tang | 202135591

Santiago Rivett Barragan | 202114556

Yizhou Fang | 202340798

Group 23

EE318 Project

Contents

1 Executive Summary

This report details the design choices, progress, and operation of components that form a Mini-grid game. Such a project idea was selected as it satisfies the design brief by incorporating an MSP430 microcontroller to drive the game and a Printed Circuit Board (PCB) to handle the hardware providing a demonstration that lasts around 10 minutes. The demonstration aims to provide the users with a fun and interactive experience that sheds light on United Nation's Sustainable Development Goal 7 - Clean and Affordable Energy by harnessing the main themes of a powergrid [1]. The group designed the game to be educational to the users and audience by attempting to highlight a real-world engineering problem that is access to clean and affordable energy in developing countries.

The demonstration consists of 2 players with Player 1 handling the Generation element and Player 2 handling the Consumption element. Player 1 uses an analogue joystick to capture light from the LED arc to generate power. This power is sent to Player 2 who controls where the power is directed.

2 Description of Project Objective

The group was tasked with the creative design of an interactive demonstration tailored for a young audience. This experience should be user-friendly in setup and educational, aiming to engage and provide informative content to the audience throughout. The primary goal is introducing them to the world of engineering, with hopeful ambition of aspiring the next generation of engineers.

The demonstration was constrained by the following main criteria:

1. Align with Sustainable Development Goal 7 - Clean and Affordable energy
2. Utilise the MSP430 microcontroller as the main driver
3. Include a working Printed Circuit Board (PCB)
4. Provide a demonstration that lasts around 10 minutes
5. Adhere to the £20 project budget

Secondary and tertiary aims included amelioration of project management skills and teamwork, as well as technical report writing - all of which will be integral to the groups professional engineering careers.

With these objectives in mind, the group decided to implement a Mini-grid game that simulates a time period in a developing Sub-Saharan African country with the intentions of highlighting the difficulty of renewable energy access in developing countries - described by SDG Target 7.b [1].

3 System Design

This section of the report details the breakdown of the project into key subsections. The design process will be supported by thorough explanations of both hardware and software component operations.

3.1 System View

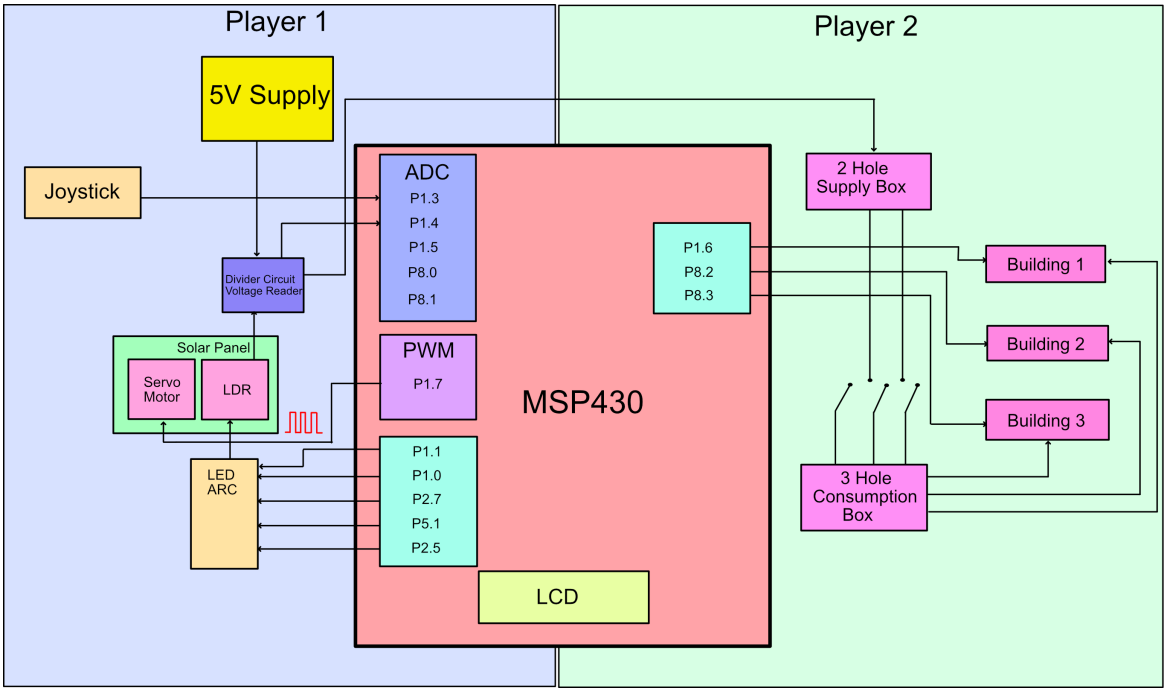

Figure 1: System Block Diagram

Figure 1 captures the main high-level themes of the entire system design, describing the interactions between both Player 1 and Player 2 sides with the MSP430 microcontroller. It also details the low-level interactions - in particular, the exchange between the MSP430 ADC peripheral, voltage reader component, and analogue joystick input.

General game operation commences on the Generation side with the LED arc being driven by pins P1.1, P1.0, P2.7, P5.1, and P2.5. The light from these LEDs stimulate a change in resistance across the LDR terminals which subsequently creates a change in voltage inputted into the voltage reader. Note that Player 1 controls the position of the LDR by use of an analogue joystick, coupled with the PWM pin P1.7, to drive the servo motor. The voltage is read into P1.4 where a score for Player 1 is calculated and presented on the LCD screen upon request.

Taken from the voltage reader on the Player 1 side, the 2 Hole Supply box is supplied with voltage and needs to be transmitted to different buildings through banana cable connections into the 3 Hole Consumption box. Each building has an associated indication LED which is driven by P1.6, P8.2, and P8.3, which are also configured to read voltages from the 3 Hole Consumption Box. The LED indication subsystem offers a dynamic means of communicating progress to the user by increasing the flashing rate until task completion.

3.2 Player 1 - Generation

The following sub-sections follow the software and hardware design of components that contribute to Player 1's operation.

A 3D-printed arc with five LEDs on the interior simulates the sun's path by randomly lighting up the LEDs in sequence to test the users reflexes. The solar panel movement is controlled by an analogue joystick which is guided by the user with the aim to capture as much sunlight as possible. The aim of this implementation is to create a fun demonstration which will fully engage the user.

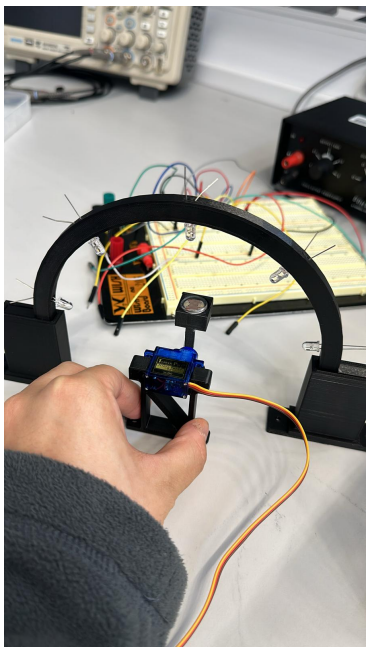

Figure 2: Player 1 Generation Visualisation

3.2.1 Servo Motor Control

Upon research, the DC motor that best suited the design critera was the SG90 Servo Motor. This motor supplied the complete 180° rotation that was required in the solar panel design. The movement of the servo motor was controlled by Pulse Width Modulation (PWM). This mechanism involves manipulating duty cycle of a signal. A visual aid is provided below in Figure 3.

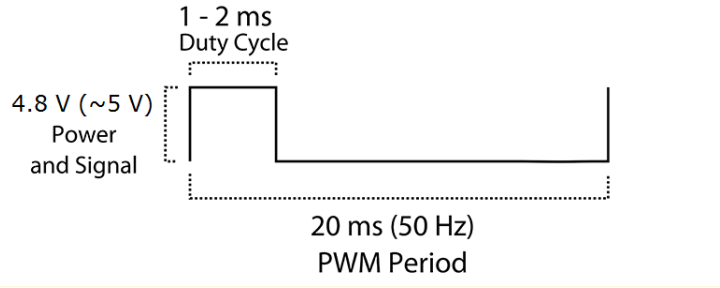

Figure 3: Servo motor control

Both Pulse Width Modulation (PWM) and duty cycle are used to control the movement of the motor but both operate very differently.

Pulse Width Modulation is typically used to control the average power delivered to a load by varying the width of the pulse. To sum up concisely, PWM is used to control the position of the servo motor and the width of the pulse will determine the position of the motors shaft [2].

As for the duty cycle, this refers to the time that the signal is binary high compared to the total period of the signal. By programming the duty cycle of the PWM, the servo motor can be precisely controlled.

Initially, the PWM signal was measured manually by operation of an oscilloscope. These measurements gave the exact PWM and duty cycles required in order to control the motor as required.

With careful guidance from the **MSP430FR4xx User Guide**, the servo motor was implemented to behave exactly as intended through manipulation of specific registers. In particular, timer registers played a crucial role in the successful operation of the servo motor [4].

The group encountered a minor issue regarding the starting position of the servo motor arm. Ideally, the arm would sit in a 90° position from the start and shift ± 90 degrees to reach first and last LEDs. This issue can be better understood by looking at Figure 2.

Minor issues existed with this design that were deemed too significant to prioritise. Namely, the servo motor arm did not rotate a full 180° but rather 160° when initialised at a fully upright starting position. This issue was quickly solved by offsetting the arm slightly to the right.

3.2.2 Analogue Joystick

In partnership with the servo motor, an analogue joystick was successfully programmed and integrated into the project to manually control the movement of the motor. The particular model used was the 'Parallax 2 Axis Joystick'.

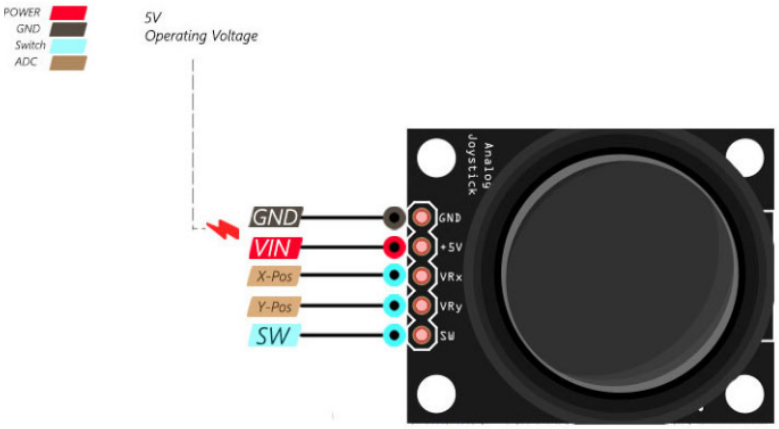

Figure 4: Joystick Data sheet [3]

The analogue joystick, as shown above in Figure 4, can provide up to 3 separate controls (VR_x,VR_y and SW) but for this project only movement in the x axis is required.

ADC (Analog-to-Digital Converter) configurations are crucial for linking the joystick to the servo motor. It enables the translation and mapping of the joystick's ADC position to a digital signal that can be processed by the microcontroller which is then used to alter the motor's position, this is achieved with techniques discussed above.

With aid from the EE312 Semester 1 material on ADC connections, as well as the User Guide, an ADC connection was successfully made between the joystick and servo motor. Figure 5 displays the connections made.

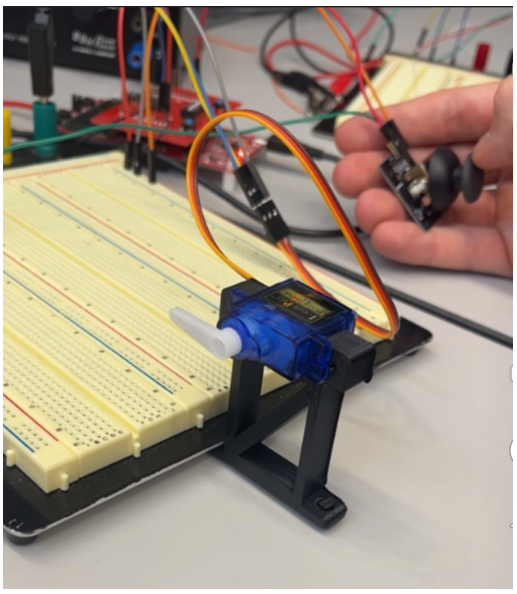

Figure 5: motor and joystick

The steps to configure an ADC are as follows:

1. Configure a pin suitable for analog conversions
2. Configure clock source and operation modes
3. Configure ADC mux as well as positive and negative references
4. Configure interrupt

The most crucial functions created were the *analogread* and *map* functions. The *analogread* function takes in a joystick input value and triggers an ADC conversion. This conversion would then be translated into a set of linear servo motor positions with use of the *map* function. Listing 1 showcases the described code implementation.

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min)
    + out_min;
}

int analogRead()
{
    ADCCTL0 |= ADCENC | ADCSC; // Start conversion
    while (ADCCTL1 & ADCBUSY); // Wait for conversion to complete
    return ADCMEM0; // Return conversion result
}
```

Listing 1: Crucial functions for ADC connections

Using the code listing above, as well as other crucial functions, the joystick and motor worked in perfect harmony to complete the most important sub component of Player 1's operation.

3.2.3 LED Arc

The LED Arc was designed to mimic the sun and does so by flashing one of 5 LEDs at a time. Its semi-circular nature means that each LED is equidistant from the LDR positioned at the centre of the LED arc. Such a configuration means that the LEDs are located at 0°, 45°, 90°, 135°, 180° relative to the centre. The LED arc is elevated by 2 stands which are 5cm tall - these help to align the LDR with the centroid of the semi-circular arc.

A wide array of LED colours were thoroughly tested before finalising the selection for the arc design. Parameters like voltage and resistor pairings, brightness, and availability were used as metrics for selection. Ultimately, the white LED was drafted as the final candidate due to its intense brightness and acceptable resistor-voltage pairing.

The final LED arc design can be visualised in Figure 6 below.

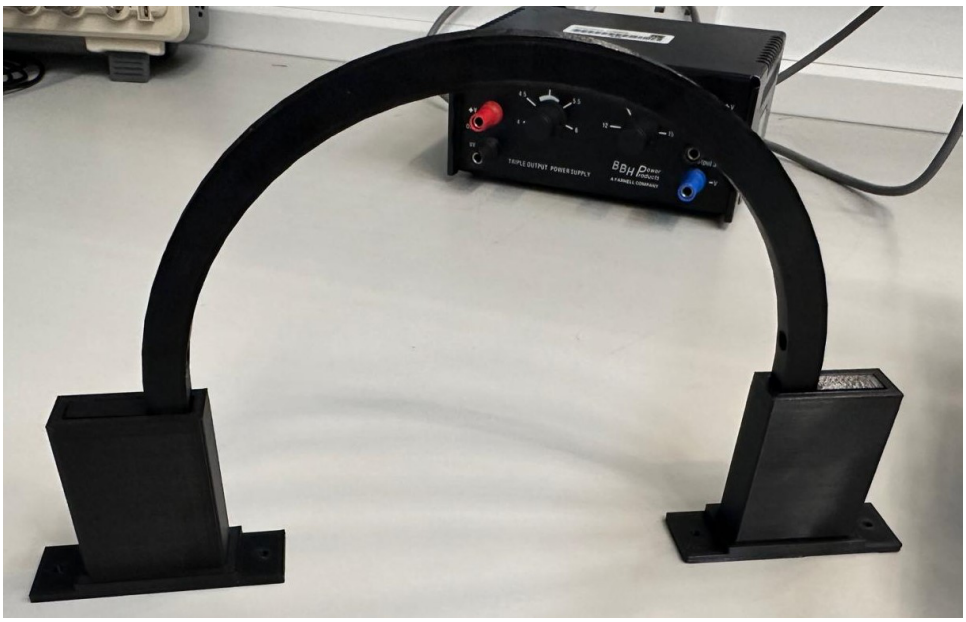

Figure 6: 3D printed LED arc

3.2.4 LDR solar panel configuration

Initially, the *Solar Panel* was designed using a photodiode, with the MSP430 reading the corresponding light intensity. For this configuration, the MSP430 would need to serve as an ammeter, since light intensity is directly proportional to current for a photodiode.

However, during the testing phase of this component, it was discovered that the sensitivity of the photodiode was poor, and its overall performance was sub-par even when multiple photodiodes were used. Upon this realisation, the group opted for a Light Dependant Resistor (LDR) as its replacement due to its stronger sensitivity capabilities.

An LDR exhibits an inversely proportional relationship between light intensity and resistance which is displayed in (1).

$$intensity \propto \frac{1}{V_{out}} \tag{1}$$

Hence, the following voltage divider circuit was realised to demonstrate the correlation between light intensity and voltage readings, mirroring the behaviour of a solar panel. This is shown in Figure 7.

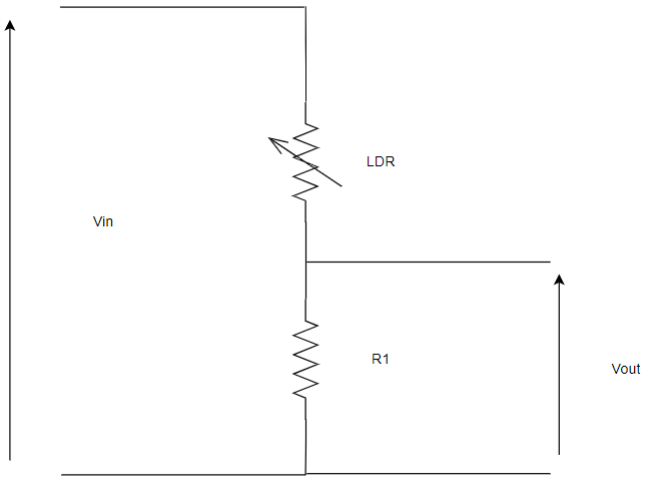

Figure 7: LDR Voltage Divider Circuit

The circuit takes advantage of the LDR properties and measures the voltage of the resistor in series with the LDR. Such a configuration works for a solar panel emulation as when the light intensity increases, the resistance drops across the LDR hence leading to a subsequent increase in voltage across resistor, R1. Likewise, when the light intensity decreases, the resistance and voltage decreases for Vout and R1. This relationship is modelled in (2).

$$intensity \propto V_{out} \tag{2}$$

A 5V source was deemed sufficient in powering this circuit as it provides a reasonable value of Vout that can be easily read by the Voltage Reader component. Furthermore, the popularity of 5V sources make them widely available online which proved to be useful when ordering a 5V plug source online for the PCB.

The voltage divider rule can be exploited in Figure 5 to yield an expression for Vout in terms of Vin, R1, and RLDR, as described in (3).

$$V_{out} = \frac{R_1}{R_1 + R_{LDR}} V_{in} \tag{3}$$

Regardless of the magnitude of Vout, the Voltage Reader code computes an algorithm to upscale this value to one that is realistic in [place of interest] and subsequently displays this on the LCD screen.

Since the magnitude of Vout was insignificant, a logical value of 1.2k was selected for R1 - as this exists in the mid range of resistor values.

3.2.5 Voltage reader

The voltage reader element is integral to the project as it enables: players to keep track of progress, functionality of the consumption indication system, and the final score calculation. It is especially important on the Player 1 side where it works in conjunction with the LDR voltage divider circuit.

This element utilises the MSP430's Analogue to Digital (ADC) converter to read varying DC voltages. The process used to setup the ADC for a voltage reader configuration can be summarised by the following process:

1. stop the watchdog timer
2. select desired ADC pins
3. set the bit conversion resolution
4. choose internal reference voltage

The watchdog timer is used to stop programs when they spiral out of control and was omitted due to low code complexity in this program. Setting the conversion resolution was crucial as this allowed the group to control the bit magnitude of the output. The internal reference voltage heavily impacts the final ADC result and can be modelled by the following equation (4) where n denotes the ADC resolution in bits:

$$ADC\,Value = Measured\,Voltage \times \frac{2^n}{Vref\,(mV)} \tag{4}$$

[5]

Through the process of trial and error, the group opted for a 10-bit resolution as this yielded reasonable values for voltage measurement. Listing 2 describes the software used to complete the ADC initialisation.

```
void configureADC(void)
{
  //begin setup
  WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
  P1SEL0 |= BIT4 | BIT5 | BIT6 | BIT7; // set ADC pins
  ADCCTL0 = ADCSHT_2 | ADCON; // ADC ON, sampling time
  ADCCTL1 = ADCSHP; // Use sampling timer
  ADCCTL2 = ADCRES; // enable 10-bit conversion results
  ADCIE = ADCIE0; // Enable ADC conversion complete interrupt
  ADCMCTL0 = ADCINCH_4 | ADCSREF_0; // set A4 (P1.4) as start,
  //internal ref voltage 1.5V
  PM5CTL0 &= ~LOCKLPM5; // Disable the GPIO power-on default
  //high-impedance mode

  __bis_SR_register(GIE); // Enable global interrupts
}
}
```

Listing 2: ADC initialisation

3.2.6 Multiple ADC Switching

One issue that arose when dealing with multiple ADC channels is that the *ADCMEM0* register can only handle one conversion a time. To overcome this, a multiple ADC switching algorithm was implemented. This algorithm polls through the ADC channels and stores the results in global variables.

After correct implementation, it was realised that the analogue joystick required more attention due to its requirement for real-time control. As a result, a new and improved algorithm was realised which is described in Figure 8.

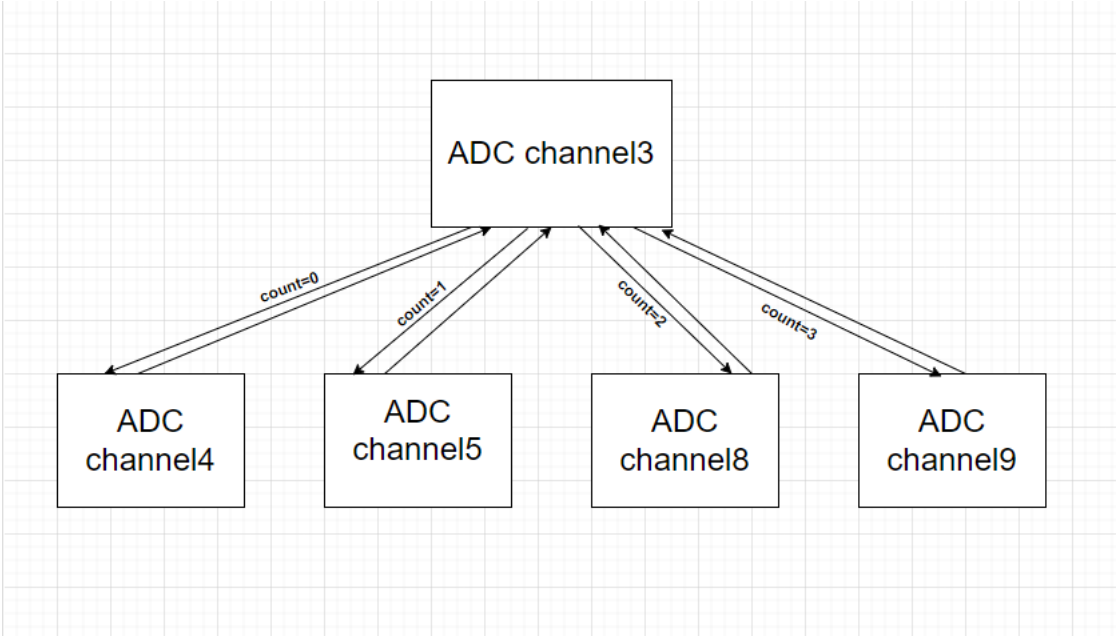

Figure 8: Multiple ADC Switching Process

A counter variable was made to track the channel of interest. The algorithm is designed to bring channel 3 to light on every second channel switch, increasing its resolution and hence making the joystick more responsive than the other voltage readers. When the channel switches to 4,5,8, the count variable increases by one and resets when it reaches channel 9 - completing the channel switching loop.

However, this has some drawbacks. Increasing the resolution on one channel significantly decreases resolution on the others. This makes the voltage reader for the solar panel slightly delayed but the group deemed this a reasonable sacrifice for finer servo motor control.

3.3 Player 2 - Transmission and Consumption

As discussed in detail above, the generated voltage from Player 1 is then directly transmitted and distributed to fulfill consumer needs. The transmission of voltage generated is controlled by player 2 with the use of 4mm banana cables. The user can join the wires to power certain appliances, some consumers may vary in required consumption in which Player 2 may need to communicate with Player 1 in increase generation or problem solve in order to decide which building to prioritise.

3.3.1 Demand indication system

To give Player 2 a sense of progress, an LED demand indication system was created. In principle, the system operates by firstly flashing the LED inside the building to indicate which building needs charged. Once Player 2 acts on this and connects power to the building, the ADC pin that is connected to the building will begin to track the total voltage count. The LED is coded to flash at 3 different rates, each of which corresponds to a different capacity level with more progress reflected by a faster flashing rate. After task completion, the LED inside the building will turn off, and another building will activate to repeat this process.

3.4 LCD display

Initially the goal for enabling demo players to gauge success was through the use of multiple external Liquid Crystal Display (LCD) screens. The LCD would show key stats like generation, consumption, and efficiency as well as time remaining.

Initially, the Keyestudio LCM1602 model was selected but this proved troublesome to implement due to software complications with the I2C communication protocol. Despite the groups best efforts, no significant progress was made and so it was decided to display all the key information on the MSP430's own LCD screen. One screen would serve as multiple displays by using the two switches on the MSP430 to toggle different stats. This design implementation can be visualised in Figure 10.

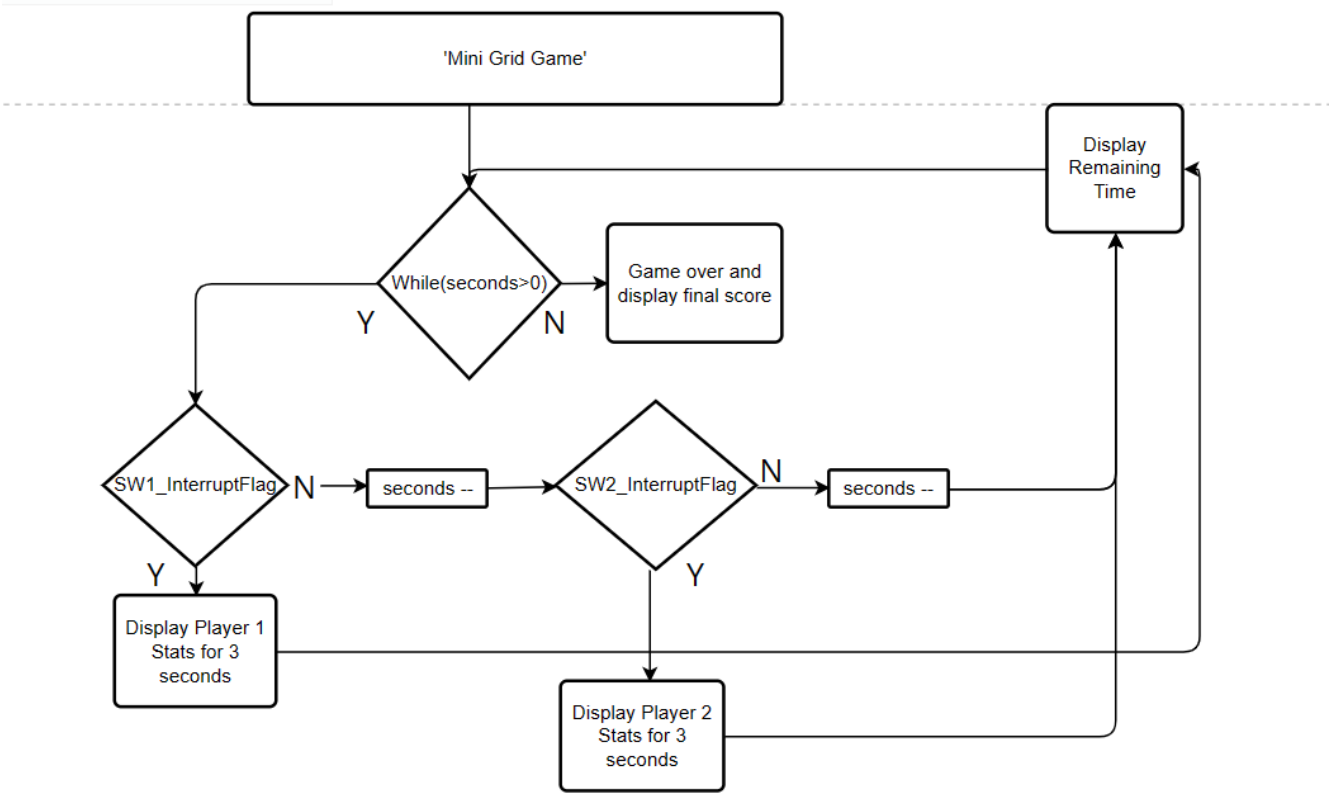

Figure 9: Flow Chart

An interrupt is defined as the automatic transfer of software execution in response to events (from hardware) that are asynchronous with the current software execution.

The switches are all controlled via interrupts, which maximise sensitivity of button presses. Its responsiveness can be characterised by placing the interrupt flag at the top of the hierarchy. This prioritises the button switch whenever it is pressed. By setting the switches as high-priority tasks driven by interrupts, it enables players to toggle freely between screens in the game without inclusion of an external LCD display.

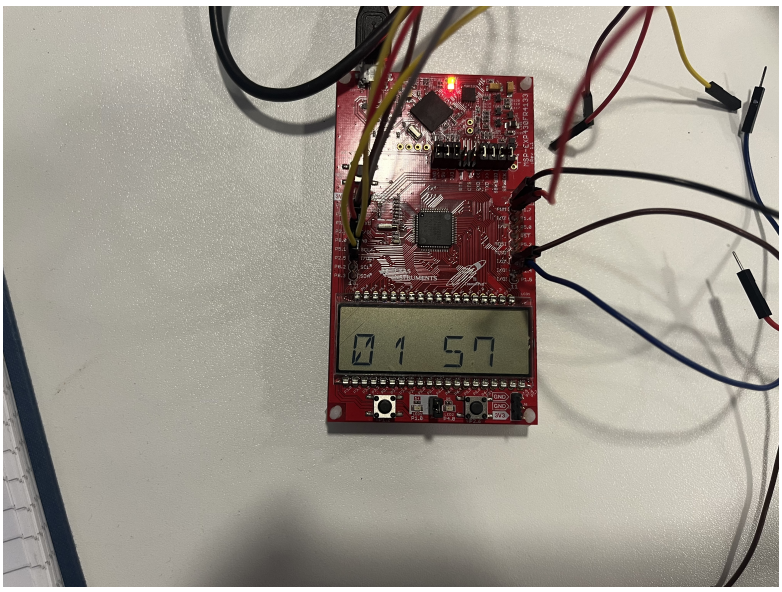

Figure 10: Countdown visulisation on MSP430

3.5 PCB design

The PCB is integral to the overall design, providing a neat and compact structure to handle circuitry. The circuit displayed in Figure 11 was realised by combining elements of individual sub-components onto a convenient viewpoint.

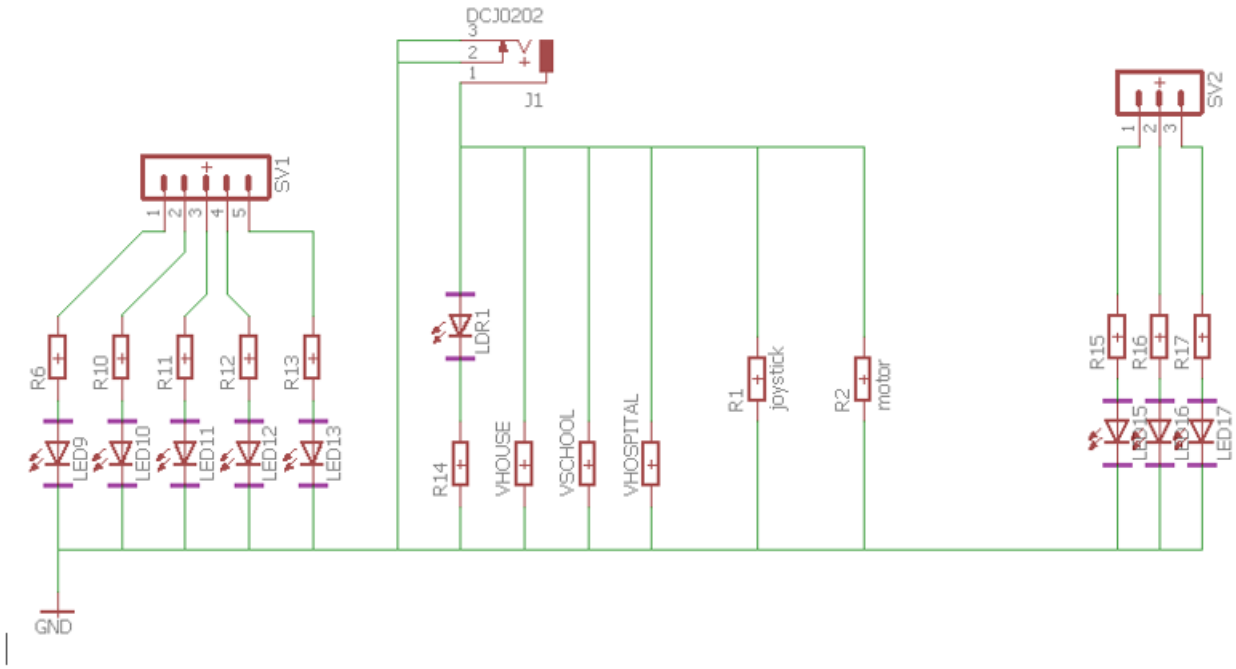

Figure 11: PCB circuit design

Table 1 lists the circuit components and the corresponding responsibilities in the overall design, where the first column conveniently details the sub-system

Sub-system	Component Name	Functionality	Value
LED arc	SV1	Header pins to enable female-female connections to MSP430	N/A
LED arc	R6, R10, R11, R12, R13	Resistors to protect LEDs of arc	150Ω
LDR Voltage Reader and Solar Panel Movement	5V Barrel Jack	Provides voltage supply for analogue joystick, servo motor, and LDR voltage divider	5V
Solar Panel Movement	R1 joystick, R2 motor	Serves as placeholders for Analogue joystick and Servo motor connections	N/A
LDR Voltage Reader	LDR	Emulates solar panel by reacting to light changes	0-10kΩ
LDR Voltage Reader	Adjacent resistor	Completes the voltage divider circuit, voltage across this resistor is directly proportional to light intensity changes	1.2kΩ
Consumption Voltage Indication	SV2	Header pins to enable female-female connections to MSP430	N/A
Consumption Voltage Indication	VHOUSE, VSCHOOL, VHOSPITAL	Resistor components placed here as placeholders for wires to track voltage	N/A
Consumption Voltage Indication	R15, R16, R17	Resistors to protect LEDs inside consumption buildings	150Ω
Consumption Voltage Indication	LED15, LED16, LED17	LEDs to indicate progress and what building needs charged	N/A
All	GND	Provides a common ground for the design, enables mutual grounding with the MSP430	0V

Table 1: PCB circuit component definitions

Upon finalisation of the circuit design, the next phase of the PCB design process entailed mapping individual components onto the final board layout to prepare it for printing on tracing paper. The group organised the layout strategically, attempting to optimise a design to simplify the soldering process.

Figure 12 captures the PCB layout submitted for printing. Copper traces existed on one side only as the group believed that it would reduce soldering complications. It was discovered that this design choice proved advantageous in the later parts of the design, where a PCB holder was created to keep the PCB secure whilst allowing connections to be made from the top.

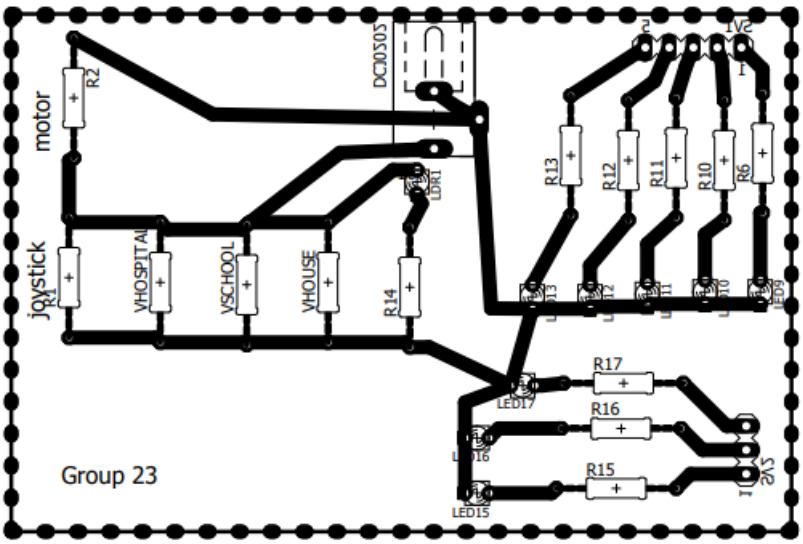

Figure 12: PCB layout

3.6 3D printing

The process of designing 3D mechanical components for the demonstration board was one that was repetitive. It involved measuring appropriate dimensions whilst factoring in some margins for error, modelling it on Solidworks software, and testing the layout with other components. First try successful prints were seldom the case so multiple iterations of designs were required most of the time. Figures 13, 14, 15, 16, 17, 18, 19, 20 showcase the sketches used in the final design.

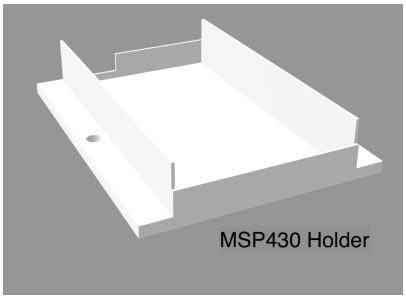

Figure 13: MSP430 Holder

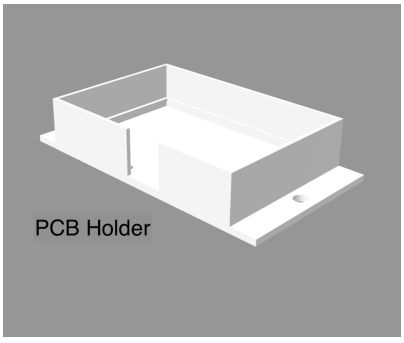

Figure 14: PCB Holder

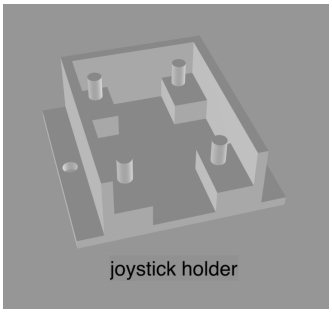

Figure 15: Joystick Holder

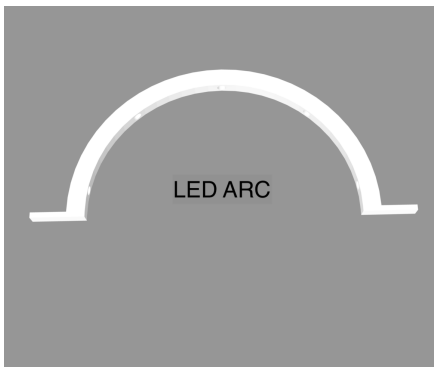

Figure 16: LED arc

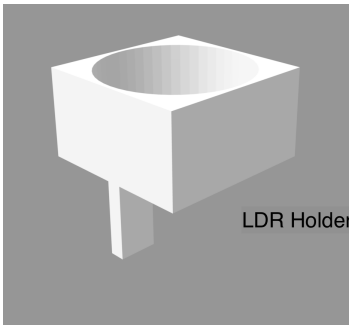

Figure 17: LDR Holder

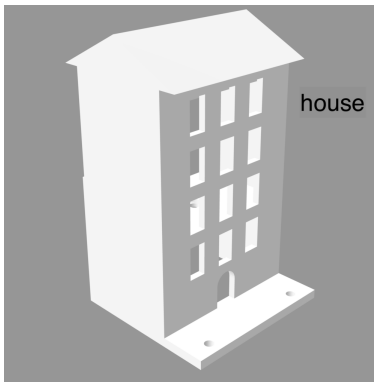

Figure 18: House

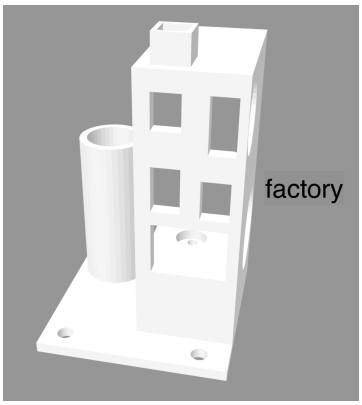

Figure 19: Factory

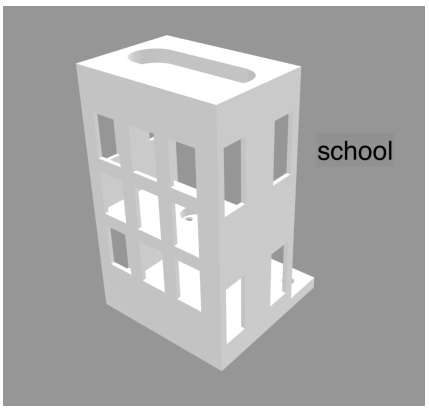

Figure 20: School

3.7 Final Mechanical Design Layout

For the final design layout the group's goal was to present all subsystems as neatly as possible, as well as limiting the amount of wires presented. Another obstacle that the group was ensuring that all demonstration items can fit into the box dimensions.

The group discovered an intelligent method of storing components. With the help of

the mechanical workshop, 2 perspex boards were used as demonstration (demo) boards. These boards have hinges to allow the bottom side to fold over and fit neatly into the box.

Having the 3D printed component structures screwed and bolted into the demo board helped to contribute to an easy and hassle-free setup. Also, rubber stands were glued to the bottom of the board to increase surface friction and keep the board stable during demonstration.

Figure 21 is attached below to act as a visual aid for the demo-board and final layout.

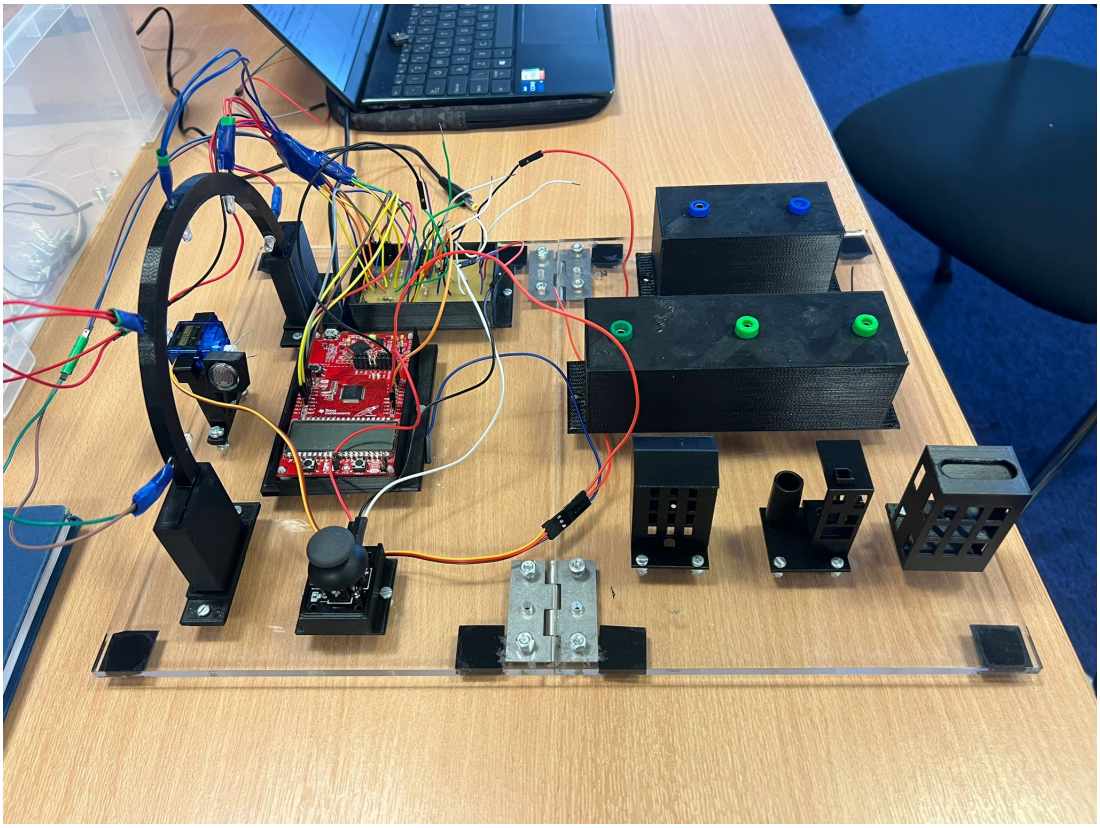

Figure 21: Demo board layout

3.8 Scoring System

A scoring system is used to provide the users a gauge of success and to add to the fun element of the game. Both Players are scored on respective performances and from this, an overall combined score is created which serves as the final team score. The score will the power generated at the Player 1 side, the amount of buildings powered at the Player 2 side, and power efficiency.

Power efficiency holds the strongest weighting on the score, as it was believed to be a

powerful teaching opportunity for both users and the audience, enhancing the appreciation and understanding for energy conservation.

It was thought that the addition of a leader board would have increased the fun element of the game but due to time constraints, and the group's focus on other subsystems, this was not implemented.

4 Results and Achievements

Figure 21 captures the main achievement of this project - a working demonstration driven by an MSP430 that uses a PCB to map connections. It provides a demonstration, lasting around 10 minutes, that is packed with educational content when played correctly in accordance with the user guide. The PCB placed in its designated holder is displayed in Figure 22.

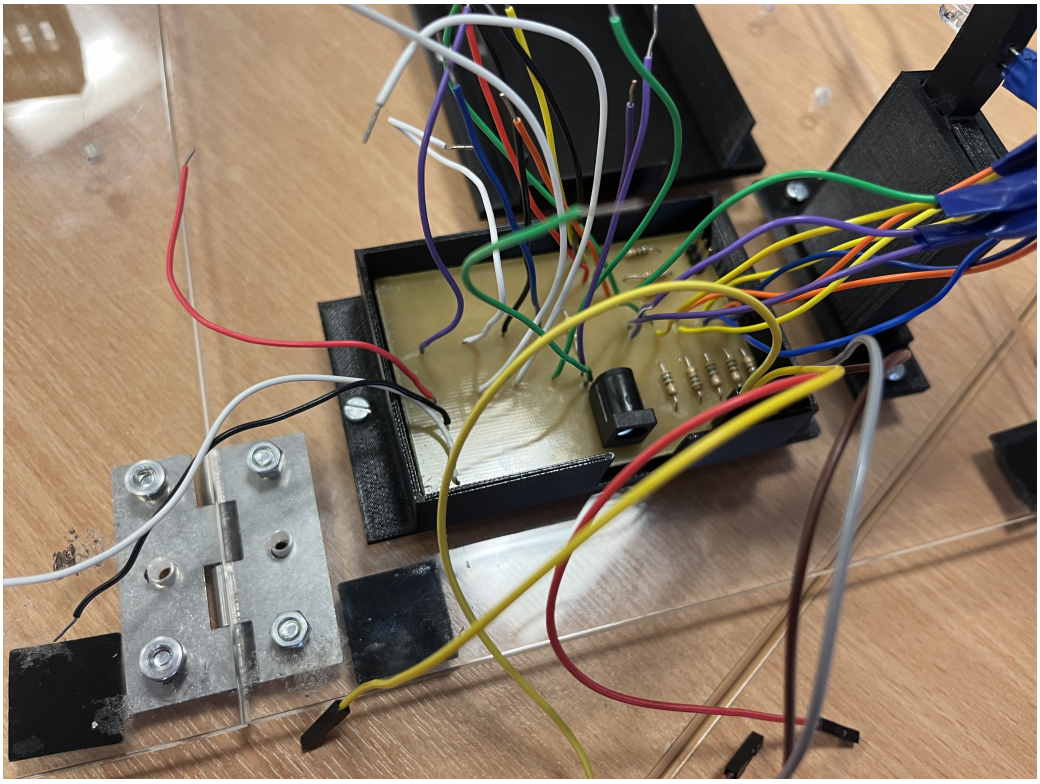

Figure 22: PCB in holder

Following the gantt chart strictly allowed the group to fully implement the Player 1 side which included a fully functioning analogue joystick, servo motor, voltage reader, LDR divider circuit, and LED arc. On the mechanical design side, a demo board was crafted to provide a clean and tidy space to play the game. Creation of this board also streamlined the set up and pack up processes.

There exists a small group of sub components that were successfully implemented but were omitted in the final design. This is due to unforeseen difficulties that exist with combining the code into a singular project file. An example of such is the sequential nature of embedded C code. One instance of this was conquered whereby the LED arc took precedence over the ADC channel switching however, this was solved after great

efforts. Integrating the LED demand indication sub system with the Player 1 code proved to be a greater challenge than imagined, due to the priority issues as mentioned previously, and subsequently lead to its exclusion in the final project demonstration. This rendered the fully operational banana box transmission system moot, contributing to the removal of Player 2's side from the project, and ultimately resulting in a 1 Player game. The issues mentioned could be resolved if the group had additional time, but the Project management provides a deeper context for the problems faced during the project.

With considerations to the group's high expectations, the group did not manage to attain the level of realism as desired. The design brief stated that the project must be interactive and the group satisfied this on all fronts by focusing on gamification. However, equally important was the attention to the educational aspect of the demonstration and the group thought this was best executed through adding elements of realism to the game. Notably, it was planned to select a place of interest and model weather trends in that region.

From a non-technical standpoint, completion of this project enhanced the project management skills of the group as taking on a project of this calibre was tied with many problems that were foreign to the group. As discussed in the Project Management section, the group gained invaluable experience dealing with and adapting to unexpected problems.

5 Discussion

5.0.1 System Analysis

A measure of project success can be evaluated by comparing what was achieved against the project design brief. Table 2 describes this.

Design Brief	Final Proposal	Successful?
Working PCB	Working PCB that maps appropriate connections to relevant sub components	Yes
Inclusion of MSP430	Working PCB that maps appropriate connections MSP430 drives entire project	Yes
Relation to SDG 7	Themed around a Mini-Grid game to raise awareness about target 7.b	Yes
Provide a 10 minute demonstration	10 minute demonstration provided	Yes
Adhere to the £20 budget	Project net spend of £0	Yes

Table 2: Outcomes vs Project Design Brief

As described above, the targets detailed in the design brief were achieved in all aspects. Although the PCB lacks major complexity, it must be remembered that the project design is software heavy and hence does not require much hardware and PCB circuitry. The PCB is deemed successful and fit for purpose as it is tailored towards project's hardware needs

After rigorous testing, it was discovered that the overall project system is robust as it worked upon start up every time without failure. It is also considered mechanically robust, since the project equipment was dropped a numerous amount of times without sustaining damage.

One positive taken from this testing phase was that the game did not crash on any occasion and withstood the input combinations tested. This included pressing down on the joystick to initiate its hidden switch.

A slight shorting error on the PCB causes the first LED at 0° to light. This means that the users can keep the servo motor locked in a fixed position to yield a maximum score. However, this is advised against and a warning will be featured in the revised user guide.

5.0.2 Strengths and Weaknesses

The strengths of the proposed design include its neat component layout and clever design. These facilitate easy setup and ease of use, indirectly making the game more enjoyable for the users. The integration of banana cables was a nice touch as it provides a convenient and safe way to make electrical connections in a live circuit. Colour coding the supply and consumption holes was considered a good idea as it offered clear instructions on connections.

To make connections between components possible, the PCB demanded male-male wires to be soldered onto its surface. Many connections were required in the design and so wire entanglement was immanent. The group noticed this issue and suppressed it by using electrical tape to round up common points, reducing the eyesore of many loose wires.

6 Further Work

Due to time constraints, and commitments to other modules, the group were unable to express their full ambitions in the final implementation. Namely, the aspect of realism faded as the project developed as the group shifted focus towards the fun element of the project. Making the project more realistic would have strengthened the project's link to SDG 7 and hence would have delivered an even more powerful message on real-world engineering problems.

Realism can be enhanced by the introduction of a battery storage to reflect the intermittent nature of solar energy. This inclusion would help users understand the importance and difficulty of energy conservation in developing countries - something that we take for granted in the modern western world.

Furthermore, simulating real-world climate patterns would add another layer of realism. This could be accomplished by dimming the LEDs within particular intervals to indicate that cloud cover can hinder solar panel performance.

Lastly, integration of different solar panel arrangements could highlight some key fundamentals of solar physics. Adding an option to select between a horizontal tilt, fixed optimal tilt, and solar tracker arrangements would provide even more entertainment for the audience while delivering an instructive lesson on the impacts of Global Tilted Irradiance (total power density acting on a surface) when installing designing a solar panel.

The suggestions for future work are not indicative of group project failure but rather a testament to the group's ambitious mindset. The criticality used in other sections personifies the groups constant pursuit of perfection.

7 Project Management

To ensure a smooth running project, the group created a Gantt chart in the early weeks of the project launch. A draft revision was required for this chart as the critical path was perceived to be excessively long. The revised draft accommodated this issue and included more tasks that occurred in parallel. Extracts from this Gantt chart are displayed in Figure 23, 24, 25 where the task completion, date, and members responsible can be visualised.

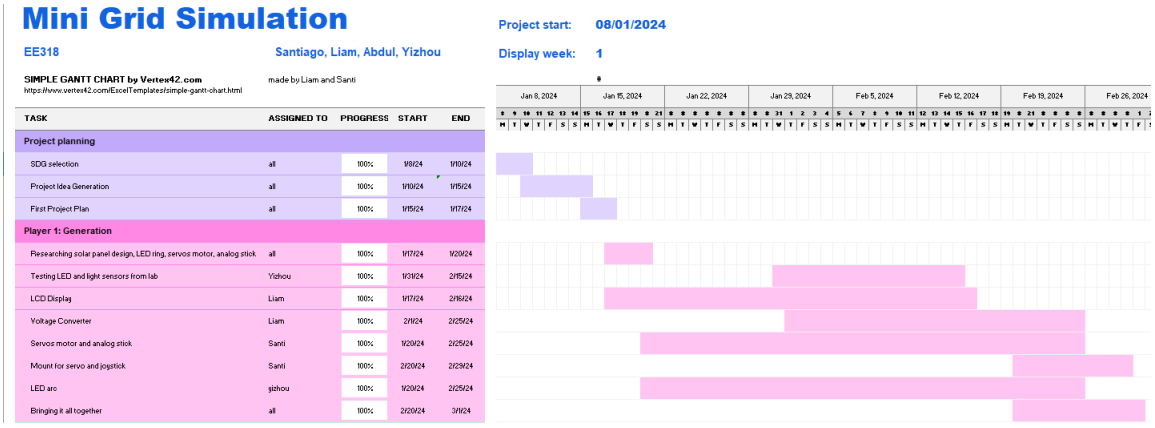

Figure 23: Gantt Chart Week 1

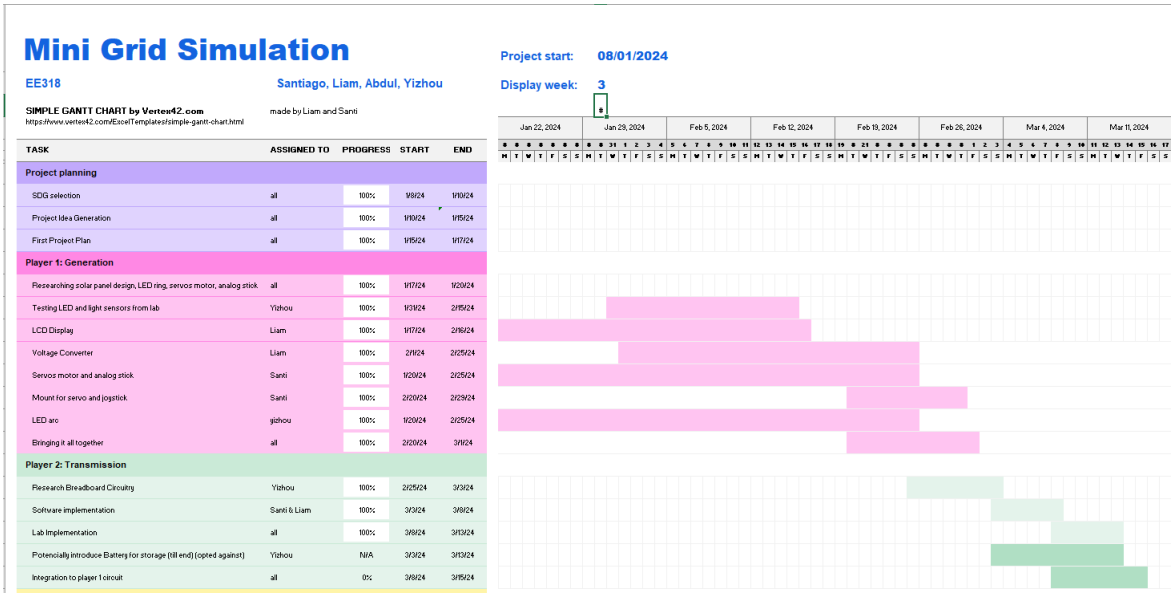

Figure 24: Gantt Chart Week 3

Additional Tasks				
Eagle modelling of PSpice simulatiuon	Santi & Liam	100%	3/2/24	3/5/24
Order PCB	all	100%	3/12/24	3/16/24
Soldering PCB	Santi	100%	3/17/24	3/20/24
Extended LCD Display (calculations ect) (opted to omit this)	Liam & Santi	N/A	3/1/24	3/13/24
Game Logistics (Scoring system, leaderboard ect)	Abdul	0%	3/1/24	3/13/24
Final Prototype design	all	100%	3/11/24	3/20/24
Instructions for setting up demo	all	100%	3/20/24	3/25/24
Final presentation delivery	all	100%	3/25/24	3/30/24

Figure 25: Gantt Chart Week 8

Abiding by the time frame of the Gantt chart proved vital as significant progress was achieved in the initial stages. However, the inactive behaviour of the former group member raised concerns. The rest of the group handled the situation well, keeping close track the former member's progress as well as raising this issue to the group mentor. Ultimately, the former member decided not to engage, despite given ample opportunity to participate. In light of this, the remaining group members took on a greater workload which hindered the rate of significant progress.

8 References

References

[1] United Nations, "Goal 7 — Department of Economic and Social Affairs," sdgs.un.org, 2023. https://sdgs.un.org/goals/goal7#targets_and_indicators

[2] Sparkfun, "Servos Explained - SparkFun Electronics," www.sparkfun.com. https://www.sparkfun.com/servos

[3] "How To Interface 2-Axis Joystick w/ Arduino [Full Guide]," Electropeak, Sep. 21, 2020. https://electropeak.com/learn/dual-axis-joystick-module-interface-with-arduino/

[4] "MSP430FR413x Mixed-Signal Microcontrollers." Accessed: May 08, 2024. [Online]. Available: https://shorturl.at/ejlRY

[5] "Texas Instruments MSP430-FR5969 12-Bit ADC Setup Guide," 2015. Accessed: May 09, 2024. [Online]. Available: https://shorturl.at/ampFJ

9 Appendices

It must be noted that some code files were taken from the EE312 lab resources and built upon to create larger libraries. Namely these were the ADC library and LCD library.

```
//including header files and libraries
#include <msp430.h>
#include "hal_LCD.h"
#include "liamsetup.h"
#include "components.h"

//declaring global variables
volatile unsigned int adcValue3 = 0; // To store ADC result from P1.3
//(for joystick)
volatile unsigned int adcValue4 = 0; // To store ADC result from P1.4
volatile unsigned int adcValue5 = 0; // To store ADC result from P1.5
volatile unsigned int adcValue8 = 0; // To store ADC result from P8.0
//(for LDR voltage reader)
volatile unsigned int adcValue9 = 0; // To store ADC result from P8.1

volatile unsigned int timer = 250; // variable to track time remaining
volatile unsigned int final_score=0; //variable to track final score

//creating variable to keep track of channel process
int count = 0;

//creating variable to track current channel
unsigned int channel = 0;

//creating variable to assign ADC input channel
volatile unsigned char currentChannel = ADCINCH_3;
// Start with channel A3 (P1.3)
```

Listing 3: fully commented main.c

```
void configureADC(void)
{
  //begin setup
  WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
  P1SEL0 |= BIT4 | BIT5 | BIT6 | BIT7; // set ADC pins
  ADCCTL0 = ADCSHT_2 | ADCON; // ADC ON, sampling time
  ADCCTL1 = ADCSHP; // Use sampling timer
  ADCCTL2 = ADCRES; // enable 10-bit conversion results
  ADCIE = ADCIE0; // Enable ADC conversion complete interrupt
  ADCMCTL0 = ADCINCH_4 | ADCSREF_0;
  // set A4 (P1.4) as start, internal ref voltage 1.5V
  PM5CTL0 &= ~LOCKLPM5;
  // Disable the GPIO power-on default high-impedance mode

  __bis_SR_register(GIE); // Enable global interrupts
}
//ADC interrupt vector
#pragma vector = ADC_VECTOR
__interrupt void ADC_ISR(void)
{
    while (timer>0) //begin timer loop
    {
        //start ADC switching routine
  if (currentChannel == ADCINCH_3)
  {
    channel = 3;
    adcValue3 = ADCMEM0;

    // Read joystick input
    int servoVal = analogRead();

    // Map joystick input to servo position
    servoVal = map(servoVal, 0, 1023, -90, 300);

    // Set the servo position
    TA0CCR1 = (servoVal * 1000 / 180) + 1000;

    // Wait for the servo to get there
    __delay_cycles(200000);
    if (count == 0)
    {
        currentChannel = ADCINCH_4; // Switch to next channel (A4)
```

```
if (count == 1)
      {
           currentChannel = ADCINCH_5; // Switch to next channel (A5)
       }

  if (count == 2) {
           currentChannel = ADCINCH_8; // Switch to next channel (A8)
       }

  if (count == 3)
  {
           currentChannel = ADCINCH_9; // Switch to next channel (A9)
       }
 }

 else if (currentChannel == ADCINCH_4)
 {
   channel = 4;
   adcValue4 = ADCMEM0; // Save the result from A4
   count++;
   currentChannel = ADCINCH_3; // Switch to next channel (A3)
 }

 else if (currentChannel == ADCINCH_5)
 {
   channel = 5;
   adcValue5 = ADCMEM0; // Save the result from A5
   count++;
   currentChannel = ADCINCH_3; // Switch to next channel (A3)
 }

 else if (currentChannel == ADCINCH_8)
 {
   channel = 8;
   adcValue8 = ADCMEM0; // Save the result from A8
   count++;
   currentChannel = ADCINCH_3; // Switch to next channel (A3)
 }
```


Listing 5: fully commented main.c

```
else if (currentChannel == ADCINCH_9)
  {
    channel = 9;
    adcValue9 = ADCMEM0; // Save the result from A9
    count = 0;
    currentChannel = ADCINCH_3; // Switch back to first channel (A3)
  }
  ADCCTL0 &= ~ADCENC;
  // Allow configuration changes by disabling the ADC
  ADCMCTL0 = (currentChannel | ADCSREF_0);
  // Update channel and reference

  // Re-enable the ADC and start the next conversion
  ADCCTL0 |= ADCENC | ADCSC;
  //LED arc code
  int x = ran_num_gen();
  LED_sel(x);
 //add delay cycles - this controls how long the LED will light for
  __delay_cycles(20000);
  __delay_cycles(20000);

  //display LDR voltage value (player 1 score)
  showChar(((adcValue8 / 1000) % 10) + 48,pos1);
  showChar(((adcValue8 / 100) % 10) + 48,pos2);
  showChar(((adcValue8 / 10) % 10) + 48,pos3);
  showChar(((adcValue8 / 1) % 10) + 48,pos4);
  showChar('K',pos5);
  showChar('W',pos6);
  final_score+=adcValue8;   //keep count of final score
  timer--;
    }
    //break out of loop when timer ends, display final score
    displayScrollText("GAME OVER");
    displayScrollText("FINAL SCORE");

    while (1) {
        showChar(((final_score / 1000) % 10) + 48,pos1);
        showChar(((final_score / 100) % 10) + 48,pos2);
        showChar(((final_score / 10) % 10) + 48,pos3);
        showChar(((final_score / 1) % 10) + 48,pos4);
    }
}
```

```
1
2  int main(void)
3  {
4  WDTCTL = WDTPW | WDTHOLD;         //stop watch dog timer
5
6      //initialising components
7      Init_LCD();               //initialise LCD
8      //initialiseADC_Easy();
9
10     //initialise servo motor and joystick adc
11     configureADC();
12     configureMotor();
13
14     P1SEL0 |= BIT1;
15     PM5CTL0 &= ~LOCKLPM5;
16     //more adc setup
17     ADCCTL0 |= ADCSHT_2 | ADCON;   // ADCON, S&H=16 ADC clks
18     ADCCTL1 |= ADCSHP;             // ADCCLK = MODOSC; sampling timer
19     ADCCTL2 |= ADCRES;            // 10-bit conversion results
20     ADCMCTL0 |= ADCINCH_9;       // A1 ADC input select; Vref=AVCC
21
22     //display game introduction
23     displayScrollText("MINI GRID GAME");
24     LCD_E_selectDisplayMemory(LCD_E_BASE, LCD_E_DISPLAYSOURCE_MEMORY);
25       while(1)
26     {
27
28         ADCCTL0 |= ADCENC | ADCSC;   // Sampling and conversion start
29
30     }
31 }
32
```

Listing 7: fully commented main.c

```
1  //note: the following file was expanded from the code given
2  // in the EE312 lab notes
3
4
5  /*************************************************
6   * ADC.c
7   * Driver for ADC dial in middle of daughter board
8   *
9   * Copyright 2015 University of Strathclyde
10  *
11  *
12  *************************************************/
13
14 #include "liamsetup.h"
15 #include "hal_LCD.h"
16 #include <time.h>
17 #include <stdlib.h>
18
19 //volatile unsigned int timer = 100; // time left for game
20
21
22 volatile unsigned int adcValue = 0;
23
24
25 //taken from semester notes, maps ADC value to a servo motor position
26 long map(long x, long in_min, long in_max, long out_min, long out_max)
27 {
28     return (x - in_min) * (out_max - out_min) /
29     (in_max - in_min) + out_min;
30 }
31 //ADC initialisation - taken from semester notes
32 void setupadc()
33 {
34     //Step 1: Pin  Select P8.1 for ADC function
35
36     GPIO_setAsPeripheralModuleFunctionOutputPin
37     (GPIO_PORT_P8, GPIO_PIN1, GPIO_PRIMARY_MODULE_FUNCTION);
38
39      P1SEL0 |= BIT1;
40
41    //Step 2: Configure CLK source
42
```

```
1
2
3       ADCCTL0 |= ADCSHT_2 | ADCON;
4       // Sample and hold time, ADC on, 16 ADCCLK cycles
5       ADCCTL1 |= ADCSSEL_2|ADCSHP| ADCCONSEQ_0;
6       // Clock source: SMCLK, Clock divider: 1, Pulse-mode
7       ADCCTL2 |= ADCRES;     // Resolution: 10-bit
8       ADCMCTL0 |= ADCINCH_9;   // Select A9 for ADC input channel
9
10      // Enable ADC interrupt
11
12      //ADCIE |= ADCIE0;  // Enable ADC interrupt for channel 0
13    // __enable_interrupt();     // Enable global interrupts
14  }
15  // ADC ISR
16
17  //function to set up the motor
18  void configureMotor ()
19  {
20       WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
21
22       //initialisation the PWM P1.7 pin
23       GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN7);
24       // PWM pin on MSP430
25       P1SEL0 |= BIT7;
26
27       TA0CCR0=20000;      //PWM 20ms
28       TA0CCR1=1500;       //duty cycle of 1.5ms
29       TA0CCTL1 |= OUTMOD_7;  //timer from the user guide
30       TA0CTL = TASSEL_2 + MC_1;
31  }
32  //takes ADC value to convert it into a servo motor position
33  void SetServoPosition(int x)
34  {
35      //formula derived from trial and error
36      int duty_cycle = (x * 1000 / 180) + 1000;
37      TA0CCR1=duty_cycle;
38      __delay_cycles(200000000);
39  }
```


Listing 9: fully commented project.c

```
//converts analogue signal
int analogRead()
{
    ADCCTL0 |= ADCENC | ADCSC; // Start conversion
    while (ADCCTL1 & ADCBUSY); // Wait for conversion to complete
    return ADCMEM0; // Return conversion result
}
// TimerA UpMode Configuration Parameter
Timer_A_initUpModeParam initUpParam_A1 =
{
    TIMER_A_CLOCKSOURCE_ACLK,               // ACLK Clock Source
    TIMER_A_CLOCKSOURCE_DIVIDER_1,          // ACLK/1 = 32768Hz
    0x2000,                                 // Timer period
    TIMER_A_TAIE_INTERRUPT_DISABLE,   // Disable Timer interrupt
    TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE , // Disable CCR0 interrupt
    TIMER_A_DO_CLEAR                        // Clear value
};
Timer_A_initCompareModeParam initCompParam =
{
    TIMER_A_CAPTURECOMPARE_REGISTER_1,        // Compare register 1
    TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE, // Disable Compare
    //interrupt
    TIMER_A_OUTPUTMODE_RESET_SET,             // Timer output mode 7
    0x1000                                    // Compare value
};
//alternate way to set up ADC
void initialiseADC_Easy()
{

     GPIO_setAsPeripheralModuleFunctionOutputPin
     (GPIO_PORT_P8, GPIO_PIN1, GPIO_PRIMARY_MODULE_FUNCTION);

      //Initialize the ADC Module
    /*
     * Base Address for the ADC Module
     * Use Timer trigger 1 as sample/hold signal to start conversion
     * USE MODOSC 5MHZ Digital Oscillator as clock source
     * Use default clock divider of 1
     */
    ADC_init(ADC_BASE,
        ADC_SAMPLEHOLDSOURCE_2,
        ADC_CLOCKSOURCE_ADCOSC,
```

```
1   ADC_enable(ADC_BASE);
2
3       ADC_clearInterrupt(ADC_BASE,
4               ADC_COMPLETED_INTERRUPT);
5
6
7       ADC_enableInterrupt(ADC_BASE,
8               ADC_COMPLETED_INTERRUPT);
9
10      //Configure Memory Buffer
11      /*
12       * Base Address for the ADC Module
13       * Use input A9 POT
14       * Use positive reference of AVcc
15       * Use negative reference of AVss
16       */
17      ADC_configureMemory(ADC_BASE,
18          ADC_INPUT_A9,
19          ADC_VREFPOS_AVCC,
20          ADC_VREFNEG_AVSS);
21
22      //Start a single measurement
23      ADC_startConversion(ADC_BASE,
24                          ADC_REPEATED_SINGLECHANNEL);
25
26
27      // TimerA1.1 (125ms ON-period) - ADC conversion trigger signal
28      Timer_A_initUpMode(TIMER_A1_BASE, &initUpParam_A1);
29
30      //Initialize compare mode to generate PWM1
31      Timer_A_initCompareMode(TIMER_A1_BASE, &initCompParam);
32
33      // Start timer A1 in up mode
34      Timer_A_startCounter(TIMER_A1_BASE,
35          TIMER_A_UP_MODE
36          );
37
38      // Delay for reference settling
39      __delay_cycles(300000);
40  }
```


Listing 11: fully commented project.c

```
/**
Initialise ADC
Directly using registers
*/
void initialiseADC_Advanced()
{
  SYSCFG2 |= ADCPCTL9;               // Configure ADC A9 pin

  // Configure ADC10
  ADCCTL0 |= ADCSHT_2 | ADCON;     // ADCON, S&H=16 ADC clks
  ADCCTL1 |= ADCSHP;               // ADCCLK = MODOSC; sampling timer
  ADCCTL2 |= ADCRES;              // 10-bit conversion results
  ADCMCTL0 |= ADCINCH_9;          // A9 ADC input select; Vref=AVCC

  ADCIFG &= ~0x01;  //Clear interrupt flag

  ADCIE |= ADCIE0;   // Enable ADC conv complete interrupt
}

void displayLCD(int score) {

    showChar(((score / 1000) % 10) + 48,pos1);
    showChar(((score / 100) % 10) + 48,pos2);
    showChar(((score / 10) % 10) + 48,pos3);
    showChar(((score / 1) % 10) + 48,pos4);
    showChar('K',pos5);
    showChar('W',pos6);
    }
    //function to reset LED - part of LED arc sub component
void reset_LED() {
    P1DIR &= ~BIT1;
    P1OUT &= ~BIT1;

    P1DIR &= ~BIT0;
    P1OUT &= ~BIT0;

    P2DIR &= ~BIT7;
    P2OUT &= ~BIT7;

    P5DIR &= ~BIT1;
    P5OUT &= ~BIT1;

```

```
1
2     P2DIR &= ~BIT5;
3       P2OUT &= ~BIT5;
4   }
5   //rnadom number generator to make LEDs light in a random way
6   int ran_num_gen() {
7       srand(time(NULL));
8       // Initialization, should only be called once.
9       int r = 1 + (rand() % 5);
10      // Returns a random integer between 6 and 1
11
12      return r;
13  }
14
15  //chooses which LED lights depending on random numbers generated
16  void LED_sel(int r) {
17
18      //initialising pins
19      reset_LED();
20
21      //resetting pins - ensuring that they are off before lighting
22      // an LED (we only want 1 on at once)
23  if (r == 1) {
24          P1DIR |= BIT1;              //setting p1.4
25          P1OUT |= BIT1;              //setting p1.4
26      }
27
28      if (r == 2) {
29          P1DIR |= BIT0;              //setting p1.4
30          P1OUT |= BIT0;              //setting p1.4
31      }
32
33      if (r == 3) {
34          P2DIR |= BIT7;              //setting p1.5
35          P2OUT |= BIT7;              //setting p1.5
36      }
37
38      if (r == 4) {
39          P5DIR |= BIT1;              //setting p1.6
40          P5OUT |= BIT1;              //setting p1.6
41      }
```

```
if (r == 5) {
        P2DIR |= BIT5;              //setting p5.0
        P2OUT |= BIT5;              //setting p5.0
    }
}
//code that executes the LED arc process
void LED_ARC()
{
    while(1)
    {

            int x = ran_num_gen();
            LED_sel(x);

           //add delay cycles - this controls how long the
           //LED will light for
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
            __delay_cycles(500);
        }
}


```

Listing 14: fully commented project.c

```
1  //taken from the EE312 semester 1 resources, built on top of this
2  // .h file by adding functions specific to the project
3
4  /**************************************************
5   * ADC.h
6   * Drives ADC
7
8   **************************************************/
9  #include <msp430.h>
10 #include <driverlib.h>
11 extern void initialiseADC_Easy();  //Initialise ADC
12 // setting GPIO parameters
13 extern void initialiseADC_Advanced();
14 void configureMotor();
15 void setzeroposition();
16 void setupadc();
17 void SetServoPosition(int x);
18 long map(long x, long in_min, long in_max, long out_min,
19 long out_max);
20 void joystickRead(unsigned int *xAxis);
21 int analogRead();
22 void displayLCD(int score);
23 void LED_ARC();
24 void reset_LED();
25 int ran_num_gen();
26 void LED_sel(int r);
27 //extern void setLedDial(unsigned char value); //Set dial value
28 //extern void refreshLedDial();
29 //Refresh the display
```

Listing 15: fully commented project.h

```c
#include "hal_LCD.h"
#include "string.h"

// LCD memory map for numeric digits
const char digit[10][2] =
{
    {0xFC, 0x28},  /* "0" LCD segments a+b+c+d+e+f+k+q */
    {0x60, 0x20},  /* "1" */
    {0xDB, 0x00},  /* "2" */
    {0xF3, 0x00},  /* "3" */
    {0x67, 0x00},  /* "4" */
    {0xB7, 0x00},  /* "5" */
    {0xBF, 0x00},  /* "6" */
    {0xE4, 0x00},  /* "7" */
    {0xFF, 0x00},  /* "8" */
    {0xF7, 0x00}   /* "9" */
};

// LCD memory map for uppercase letters
const char alphabetBig[26][2] =
{
    {0xEF, 0x00},  /* "A" LCD segments a+b+c+e+f+g+m */
    {0xF1, 0x50},  /* "B" */
    {0x9C, 0x00},  /* "C" */
    {0xF0, 0x50},  /* "D" */
    {0x9F, 0x00},  /* "E" */
    {0x8F, 0x00},  /* "F" */
    {0xBD, 0x00},  /* "G" */
    {0x6F, 0x00},  /* "H" */
    {0x90, 0x50},  /* "I" */
    {0x78, 0x00},  /* "J" */
    {0x0E, 0x22},  /* "K" */
    {0x1C, 0x00},  /* "L" */
    {0x6C, 0xA0},  /* "M" */
    {0x6C, 0x82},  /* "N" */
    {0xFC, 0x00},  /* "O" */
    {0xCF, 0x00},  /* "P" */
    {0xFC, 0x02},  /* "Q" */
    {0xCF, 0x02},  /* "R" */
    {0xB7, 0x00},  /* "S" */
    {0x80, 0x50},  /* "T" */
    {0x7C, 0x00},  /* "U" */
```

```
    {0x0C, 0x28},  /* "V" */
    {0x6C, 0x0A},  /* "W" */
    {0x00, 0xAA},  /* "X" */
    {0x00, 0xB0},  /* "Y" */
    {0x90, 0x28}   /* "Z" */
};
void Init_LCD()
{
    // L0~L26 & L36~L39 pins selected
    LCD_E_setPinAsLCDFunctionEx(LCD_E_BASE,
    LCD_E_SEGMENT_LINE_0, LCD_E_SEGMENT_LINE_26);
    LCD_E_setPinAsLCDFunctionEx(LCD_E_BASE,
    LCD_E_SEGMENT_LINE_36, LCD_E_SEGMENT_LINE_39);

    LCD_E_initParam initParams = LCD_E_INIT_PARAM;
    initParams.clockDivider = LCD_E_CLOCKDIVIDER_3;
    initParams.muxRate = LCD_E_4_MUX;
    initParams.segments = LCD_E_SEGMENTS_ENABLED;

    // Init LCD as 4-mux mode
    LCD_E_init(LCD_E_BASE, &initParams);

    // LCD Operation - Mode 3, internal 3.02v, charge pump 256Hz
    LCD_E_setVLCDSource(LCD_E_BASE, LCD_E_INTERNAL_REFERENCE_VOLTAGE,
    LCD_E_EXTERNAL_SUPPLY_VOLTAGE);
    LCD_E_setVLCDVoltage(LCD_E_BASE, LCD_E_REFERENCE_VOLTAGE_2_96V);

    LCD_E_enableChargePump(LCD_E_BASE);
    LCD_E_setChargePumpFreq(LCD_E_BASE, LCD_E_CHARGEPUMP_FREQ_16);
    // Clear LCD memory
    LCD_E_clearAllMemory(LCD_E_BASE);

    // Configure COMs and SEGs
    // L0 = COM0, L1 = COM1, L2 = COM2, L3 = COM3
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_0,
    LCD_E_MEMORY_COM0);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_1,
    LCD_E_MEMORY_COM1);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_2,
    LCD_E_MEMORY_COM2);
    LCD_E_setPinAsCOM(LCD_E_BASE, LCD_E_SEGMENT_LINE_3,
    LCD_E_MEMORY_COM3);
```

```
// Select to display main LCD memory
    LCD_E_selectDisplayMemory(LCD_E_BASE, LCD_E_DISPLAYSOURCE_MEMORY);

    // Turn on LCD
    LCD_E_on(LCD_E_BASE);
}

/*
 * Scrolls input string across LCD screen from left to right
 */
void displayScrollText(char *msg)
{
    int length = strlen(msg);
    int i;
    int s = 5;
    char buffer[6] = "      ";
    for (i=0; i<length+7; i++)
    {
        int t;
        for (t=0; t<6; t++)
            buffer[t] = ' ';
        int j;
        for (j=0; j<length; j++)
        {
            if (((s+j) >= 0) && ((s+j) < 6))
                buffer[s+j] = msg[j];
        }
        s--;

        showChar(buffer[0], pos1);
        showChar(buffer[1], pos2);
        showChar(buffer[2], pos3);
        showChar(buffer[3], pos4);
        showChar(buffer[4], pos5);
        showChar(buffer[5], pos6);

        __delay_cycles(200000);
    }                                47
}
```

Listing 18: hal$_L CD.c$

```
/*
 * Displays input character at given LCD digit/position
 * Only spaces, numeric digits, and uppercase letters are
 accepted characters
 */
void showChar(char c, int position)
{
    if (c == ' ')
    {
        // Display space
        LCDMEMW[position/2] = 0;
    }
    else if (c >= '0' && c <= '9')
    {
        // Display digit
        LCDMEMW[position/2] = digit[c-48][0] | (digit[c-48][1] << 8);
    }
    else if (c >= 'A' && c <= 'Z')
    {
        // Display alphabet
        LCDMEMW[position/2] = alphabetBig[c-65][0] |
        (alphabetBig[c-65][1] << 8);
    }
    else
    {
        // Turn all segments on if character is not a space, digit
        // or uppercase letter
        LCDMEMW[position/2] = 0xFFFF;
    }
}

/*
 * Clears memories to all 6 digits on the LCD
 */

```

Listing 19: hal$_L CD.c$

```
1  void clearLCD()
2  {
3      LCDMEMW[pos1/2] = 0;
4      LCDMEMW[pos2/2] = 0;
5      LCDMEMW[pos3/2] = 0;
6      LCDMEMW[pos4/2] = 0;
7      LCDMEMW[pos5/2] = 0;
8      LCDMEMW[pos6/2] = 0;
9      LCDMEM[12] = LCDMEM[13] = 0;
10 }
```

Listing 20: $\text{hal}_LCD.c$

```
#include <msp430fr4133.h>
#include "driverlib.h"

#ifndef HAL_LCD_H_
#define HAL_LCD_H_

#define pos1 4   /* Digit A1 - L4  */
#define pos2 6   /* Digit A2 - L6  */
#define pos3 8   /* Digit A3 - L8  */
#define pos4 10  /* Digit A4 - L10 */
#define pos5 2   /* Digit A5 - L2  */
#define pos6 18  /* Digit A6 - L18 */

// Define word access definitions to LCD memories
#define LCDMEMW ((int*)LCDMEM)

// Workaround LCDBMEM definition bug in IAR header file
#ifdef __IAR_SYSTEMS_ICC__
#define LCDBMEMW ((int*)&LCDM32)
#else
#define LCDBMEMW ((int*)LCDBMEM)
#endif

extern const char digit[10][2];
extern const char alphabetBig[26][2];

void Init_LCD(void);
void displayScrollText(char*);
void showChar(char, int);
void clearLCD(void);


#endif /* HAL_LCD_H_ */
```

Listing 21: $\text{hal}_LCD.c$

Group 23: User Guide

1. Introduction

This user manual will guide you through the steps required to set up the Mini-Grid Game, the game operation, and how to tidy it away. Before opening the box, ensure to read the instructions thoroughly - this will prevent wire and tape connections from breaking.

2. Background Theory

2.1 Relation to Sustainable Development Goal 7

Our assigned Sustainable Development Goal was SDG 7 – clean and affordable energy. The Mini-Grid game explores one of the SDGs main targets of universal access to modern energy and sheds light on the difficulties of accessing clean and renewable energy in developing counties – something that we take for granted in the western world. This serves as a teaching point in the demonstration, educating the users on real-world engineering problems. In essence, the game captures the main themes of a PowerGrid and aims to educate the users on the basic elements of generation, distribution and transmission. The game also taps into the engineering behind solar panel operation whereby more power is yielded when the incidence angle is equal to 0.

3. Hardware Setup

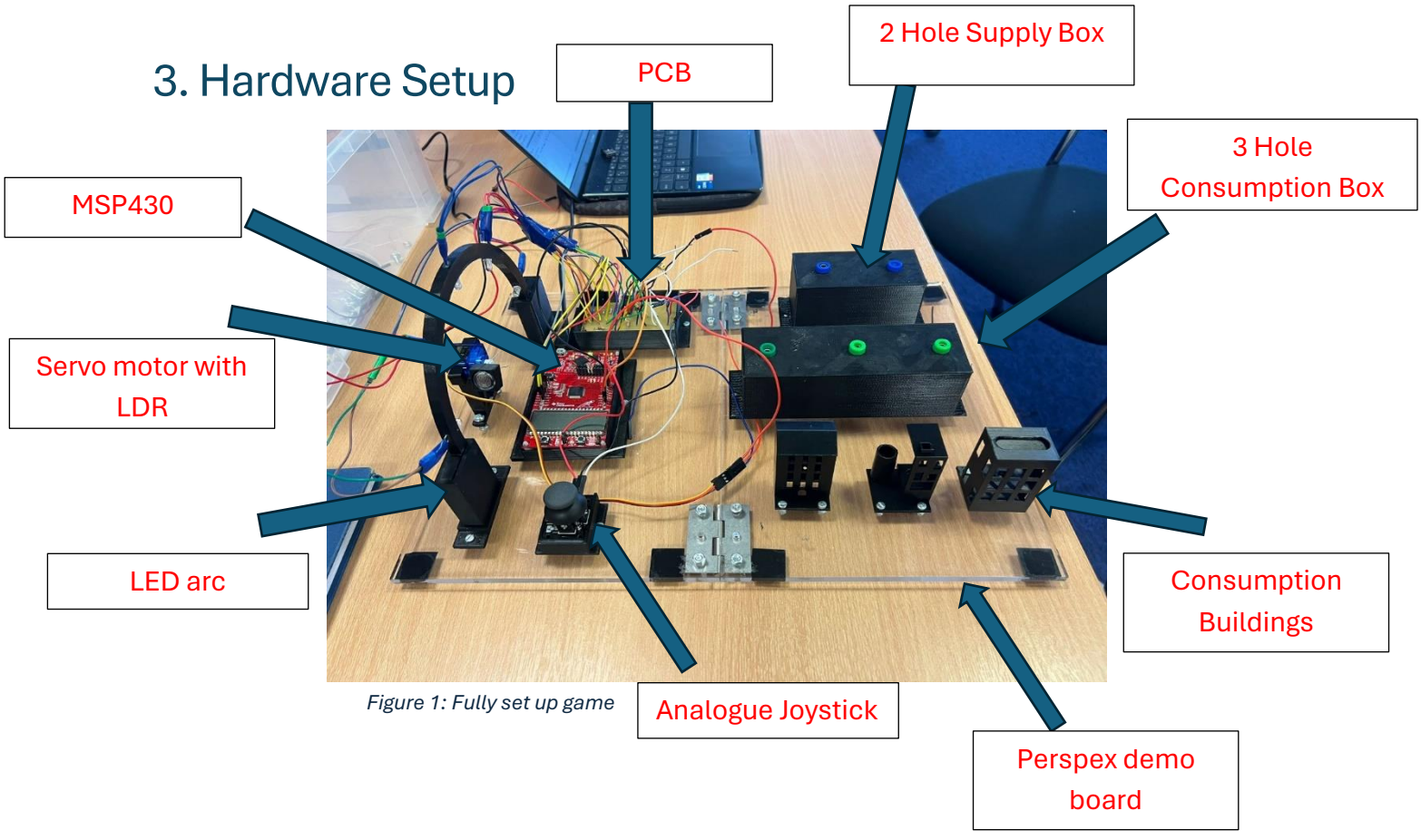

Figure 1: Fully set up game

Please read all instructions fully before plugging and powering any devices (MSP430 and plug). For any guidance on how the layout should look before commencing the game, please refer to Figure 1.

3.1 Setting up Player 1

1. Carefully, place hands on both sides of the bottom of the Perspex demo board and lift out of the box.
2. Next, unfold the bottom half and lay the entire board on a flat surface. Note: be careful of the surface that opens out as it can swing out abruptly.
3. Place the LED arc screws into their respective holes – the arc should stand upright by itself if done correctly.
4. With multiple hands, locate the side with the servo motor closest to it and tilt the board up, exposing the screws on the underside of the board.
5. Obtain the bolts from the pink bag, and screw from the underside of the board until screwed tightly.
6. Once secure, you can place the board flat on a surface again.

3.2 Setting up Player 2

1. Locate the 2 Hole Supply Box and 3 Hole Consumption Box.
2. Find the blue bag and take out the screws and bolts.
3. Line up the screws with the holes of the boxes and put them through the demo board – the boxes should be relatively stable if this is done correctly (it is worth mentioning that the wires stemming from the holes are not part of the project and should be ignored)
4. From the Player 2 side, gently tilt the board until the bottom is exposed.
5. Using the bolts obtained from the bag in Step 2, screw the bolts until they are a tight fit.
6. Locate a nearby socket and make a connection between the PCB barrel hole and the socket – ensuring that the socket is off in this step.
7. Power the MSP430 with a USB connection to a laptop.
8. Turn the socket on, run the code on the laptop provided, and enjoy the game 😊.

4. Putting it back into the box

1. Turn the socket off at the wall, unplug the USB cable from the laptop, and disconnect the plug from the barrel hole connection in the PCB.

2. By the same method used to screw the holes, unscrew the 2 Hole Supply Box and 3 Hole Consumption Box and return to its respective plastic bag.
3. Put screws and bolts back into their respective bags
4. Place the 2 Hole Supply Box and 3 Hole Consumption Box into the large transparent container ensuring that the boxes are packed flush with the container walls as shown in the diagram. A Photo is provided in Figure 2.
5. Unscrew the bolts at the bottom of the LED arc but make sure to not take the screws out (leave it as you found it). Put the bolts back to their respective coloured bag.
6. With the Player 1 Generation side facing upwards, fold the other side beneath and place into the box.
7. With the demo board sitting in the container, gently lift the LED arc and tilt it forwards such that it is nearly lying flat on the rest of the board. Figure 3 illustrates the final box layout before closing the lid.
8. Close the box lid, secure the side using the latches, and store it away.

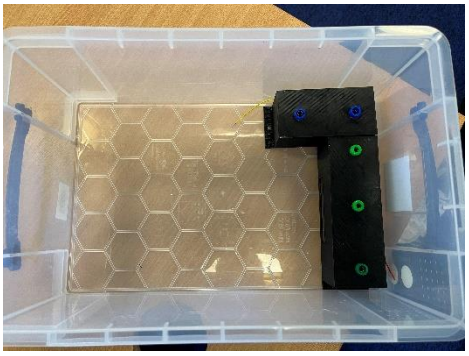

Figure 2: Supply and Consumption box layout

Figure 3: Demo fully packed away

4. Expected Results

If the steps have been correctly executed, you would expect a working Mini-Grid Game driven by an MSP430 using a working PCB to map connections. The game begins with the LED (which mimics the sun) stimulating the LDR mounted servo motor (mimicking a solar panel) which is controlled by the analogue joystick on the Player 1 Generation side. A score will be calculated based on an algorithm encoded in the MSP430 and this will be displayed on the LCD display as the game progresses – note the score sensitivity may lag due to ADC channel switching complications.

Unfortunately, the Player 2 Consumption side was not fully implemented, hence its exclusion in the final demonstration. This stemmed from several issues regarding group members and unexpected complications with the PCB.

The game will end once the internal timer reaches zero and this is indicated by a scroll text displaying “GAME OVER” before displaying the final score you achieved. You can then replay the game to try and achieve a new high score!

5.References

[1] International Energy Agency. “Key Findings – Africa Energy Outlook 2022 – Analysis.” *IEA*, 2022, www.ie (www)a.org/reports/africa-energy-outlook-2022/key-findings.