University of Southern Indiana

Pott College of Science, Engineering, and Education

Engineering Department

8600 University Boulevard

Evansville, Indiana 47712

Automatic Cornhole Scoring System

Tim Desser, Ethan Watson

ECE 491 – Senior Design

Fall 2023

Acknowledgments

There is much acknowledgement to the University of Southern Indiana and the entire engineering department for their hospitality and assistance in the pursuit of an electrical engineering degree. Many thanks to Dr. Art Chlebowski for helping find answers to all our questions about digital communication protocols and signal analysis. A special thanks to Dr. Paul Kuban for letting the group pursue this project, Mrs. Jamie Curry for her assistance in purchasing the parts and components for this project, and Steven Molinet and Derek Statz for their help with sensor testing and data interpretation on the prototype board.

Abstract

The purpose of the project is to provide an automated cornhole scoring system to give a more leisurely play as well as statistical analysis of play to the competitive players. This report includes the history and evolution of cornhole, previous attempts to assist players in keeping score, as well as the new project design to achieve the purpose previously stated. Parts and components are listed as well as reasons they were chosen. A full-size board was designed using 3D modeling and a physical board was built. Components were assembled and installed on the board. Components were tested using test code. Due to some components not operating as expected, the gameplay code was not able to run on the completed board.

Table of Contents

Acknowledgments ii
Abstract iii
Table of Contents iv
Table of Figures vii
1. Introduction 1
  1.1. History 3
  1.2. Game Play and Scoring 4
  1.3. Current Solutions 4
    1.3.1. Manual Scoring 4
    1.3.2. Remote Scoring 5
    1.3.3. Auto-Scoring with RFID 6
    1.3.4. Autoscoring with Image Recognition 8
2. Objective Statement 9
3. Specific Aims 10
4. Requirements/Constraints/etc. 10
5. Code Logic 12
6. Project Design 16
  6.1. Design Idea 16
  6.2. Systems 17
    6.2.1. Sensors 18
    6.2.2. Microcontroller 19

6.2.3. Power 20

6.2.4. Scoreboard 21

7. Game Board Design 22

7.1. Early Designs 22

7.1.1. Version 1 23

7.1.2. Version 2 24

7.2. Current Board Design 26

7.3. Hole Design 28

7.4. Whole Board Assembly 29

8. Communication 30

8.1. Within the Board 30

8.2. Sensor to Microcontroller 34

8.3. Board to Board 35

9. Testing 36

9.1. Prototype 36

9.2. Testing Plan 42

10. Results 42

10.1. Accomplishments 43

10.2. Problems Faced 44

11. What Could be Changed? 44

12. Project Planning 45

12.1. Bill of Materials 45

12.2. Timeline 47

13. Conclusion 47
References 48
Appendix A: Project Timeline 50
Appendix B: ABET Outcome 2, Design Factor Considerations 51
Appendix C: TCS3472 Datasheet 52
Appendix D: SEN0296 Datasheet 79
Appendix E: ESP32 Datasheet 84
Appendix F: E-Ink Display Datasheet 112
Appendix G: Prototype Code 182
Appendix H: Primary Communication Code 186
Appendix I: Secondary Communication Code 190
Appendix J: Gameplay Code 191
Appendix K: Button Test 204
Appendix L: Single Color Sensor Test 205
Appendix M: Multiple Color Sensors Test 206
Appendix N: Pressure Sensors Test 212
Appendix O: LED On/Off Test 215
Appendix P: LED PWM Test 215
Appendix Q: Solenoid Test 216
Appendix R: Display Test 217

Table of Figures

Figure 1.1: Game Play [2] 1
Figure 1.2: ACL Championship [2] 2
Figure 1.3: Early Cornhole Design [1] 3
Figure 1.4: Manual Cornhole Scorekeeper [4] 5
Figure 1.5: LED Remote Scoring [2] 6
Figure 1.6: RFID Scoring [3] 8
Figure 1.7: Image Recognition Scoring [4] 9
Figure 4.1: Universal Cornhole Board Dimensions [5] 11
Figure 6.1: 3D Model of Design 16
Figure 6.2: Gameplay Overview 17
Figure 6.3: SEN0296 [7] 18
Figure 6.4: TCS34752 [2] 19
Figure 6.5: ESP32 Pinout [7] 20
Figure 6.6: Lithium Ion Battery [3] 21
Figure 6.7: E-ink display 22
Figure 7.1: Board Model: Version 1 23
Figure 7.2: Board Model: Version 2 Half Section 24
Figure 7.3: Board Model: Version 3 26
Figure 7.4: Board Model: Version 3 Half Section 27
Figure 7.5: Hole Design 28
Figure 7.6: Hole Flow Logic 29
Figure 7.7: Physical Board Design 29

Figure 8.1: I2C Multiplexer Diagram 31

Figure 8.2: TCA9548A Multiplexer 32

Figure 8.3: Multiplexer Printed Circuit Board........ 32

Figure 8.4: Pressure Sensor Voltage Divider........ 33

Figure 8.5: LED Control Circuit 34

Figure 8.6: Intra-Integrated Circuit Communication 35

Figure 8.7: ESP-NOW Protocol........ 36

Figure 9.1: Prototype Sensing Red Cornhole Bag 37

Figure 9.2: Pressure Sensor Values with No Bags on Board........ 38

Figure 9.3: Pressure Sensor Values with Bag Located in Quadrant 1 38

Figure 9.4: Pressure Sensor Values of Bag Located Between Quadrant 1 and Quadrant 2 39

Figure 9.5: Color Sensor Values with No Bag on the Board........ 39

Figure 9.6: Color Sensor Values with Blue Bag Located in Quadrant 1 40

Figure 9.7: Color Sensor Values with Red Bag Located in Quadrant 1 41

Figure 9.8: Color Sensor Values with Red Bag Between Quadrant 1 and Quadrant 2 41

Cornhole Auto-Score Board

1. Introduction

Cornhole is a game played with two teams of two players. There are two boards with holes in them and each team has 4 bags. One player from each team lineups next to each board. The team that goes first is determined by a coin toss. From one board, one player from team A tosses a bag. Then, one player from team B tosses a bag onto the same board until all 4 of each team's bags have been tossed. When all the bags have been thrown for that round, points are totaled. The next round begins from the opposite board with the opposing partners taking turns tossing their bags. [1]

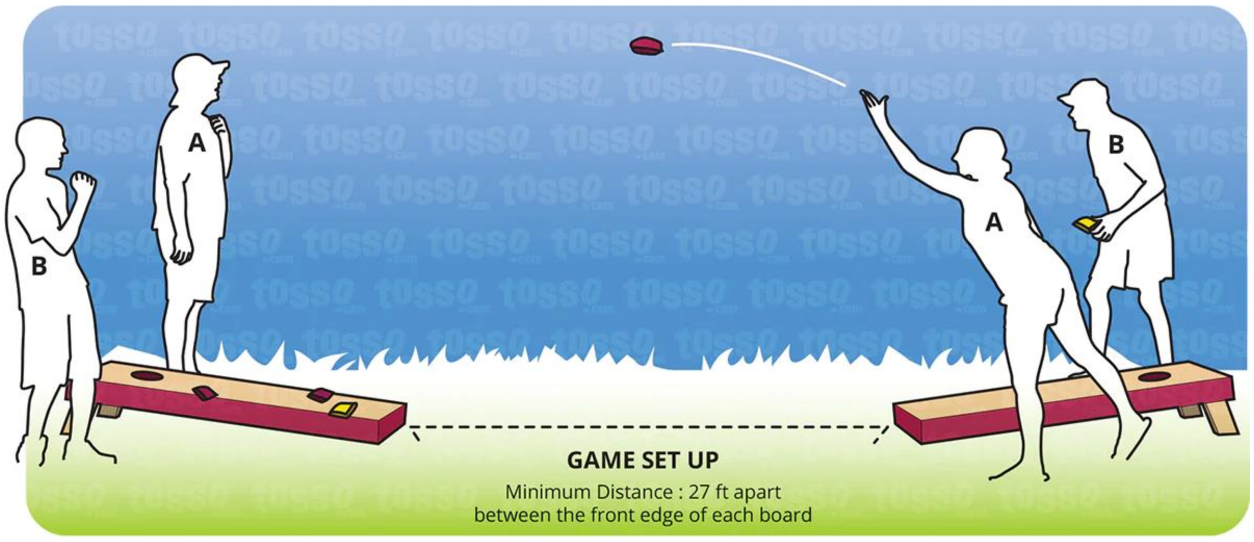

A
B
A
B
GAME SET UP
Minimum Distance : 27 ft apart
between the front edge of each board

Figure 1.1: Game Play [2]

During the upcoming of cornhole, professional leagues have been created for players that want to play competitively. Some of these leagues are American Cornhole League (ACL), American Cornhole Organization (ACO), and American Cornhole Association (ACA) are just a few of the more well-known organizations.

In 2021, there were 1024 people competing in the ACL World Championship. The winner of the tournament went home $10,000 richer.

Figure 1.2: ACL Championship [2]

Novice cornhole players may have complications when it comes to keeping the score of a match with external stimuli distracting them at any given moment. Also, Professional players need a way to accurately analyze their throws and the placement of their bags after a match to show improvement or struggles that they are encountering. Automating the scorekeeping would improve social interaction by limiting the disruption of conversation and mitigating confusion.

1.1. History

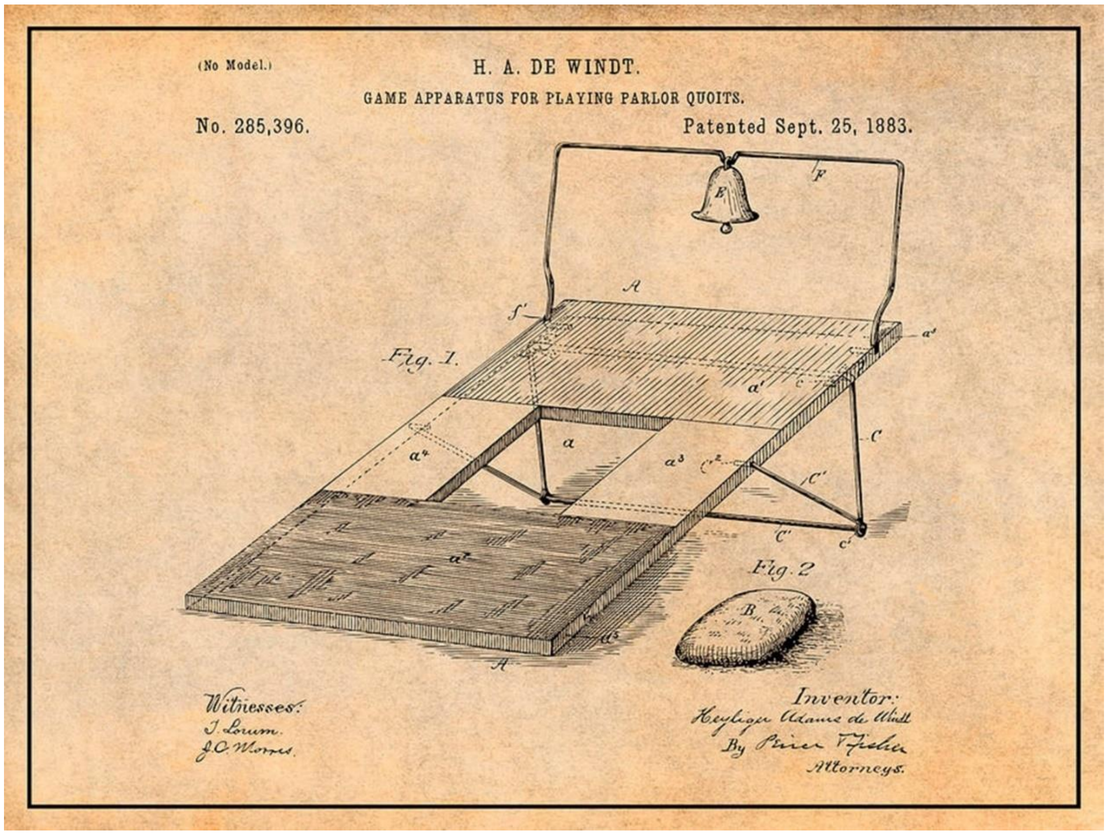


Figure 1.3: Early Cornhole Design [1]

Cornhole has many origins because no one can narrow down the invention of the backyard game. The most common stories of the origin involve many different areas. One origin involves a farmer from Kentucky, Jebediah Mcgillicuddy. It was said he invented the game just for him and his friends to play on his farm and just exploded in popularity with his townspeople. It is also rumored that cornhole stemmed from a Native American game where they would fill an animal bladder with dried beans or corn and throw them in a hole in the ground. Another origin is that a German man, Matthias Kuepermann, saw kids throwing rocks in groundhog holes and he wanted to make them a gift., so he filled burlap bags with corn that the kids could throw in a wooden box that he constructed. The most official origin is a game technically called Parlor Quoits. Heyliger de Windt got a patent for this game and his boards very closely resemble cornhole boards. Some people consider this to be the earliest model of modern cornhole boards (1883). [1]

1.2. Game Play and Scoring

Cornhole has a universal scoring system with respect to the bags. A bag on the board amounts to 1 point. A bag that goes through the hole amounts to 3 points. It also involves a cancelation method which will consider all the bags on the board and in the hole after one round and cumulate the number of points and award the resulting amount to the team with the most points in that round. [2] An easy example is if team 1 has one bag on the board and team 2 also has one bag on the board, 0 points will be awarded. Another example is if team 1 makes one bag in the hole and team 2 has one bag on the board, 2 points will be awarded to team 1.

There are two popular “modes” of cornhole played across the globe, bust or no bust. With the bust mode, a team must get 21 points to win. If that team exceeds 21 points, they are reset back down to 13 or 15 points, depending on that group’s preference. No bust is where the first team to reach a score of 21 or more wins.

1.3. Current Solutions

There are several approaches to the cornhole scorekeeping problem. These breakdown into four categories; manual scoring, remote scoring, auto-scoring with RFID, and auto-scoring with image recognition. The most common examples are discussed below.

1.3.1. Manual Scoring

A simple approach to on-board scoring is to use friction or magnetic markers that are moved manually after each round as seen Figure 1.4. This system allows players to keep track of the score between rounds. It is easy to use and requires no external power. However, this requires players to calculate the score and remember to change the score after each round. [3]

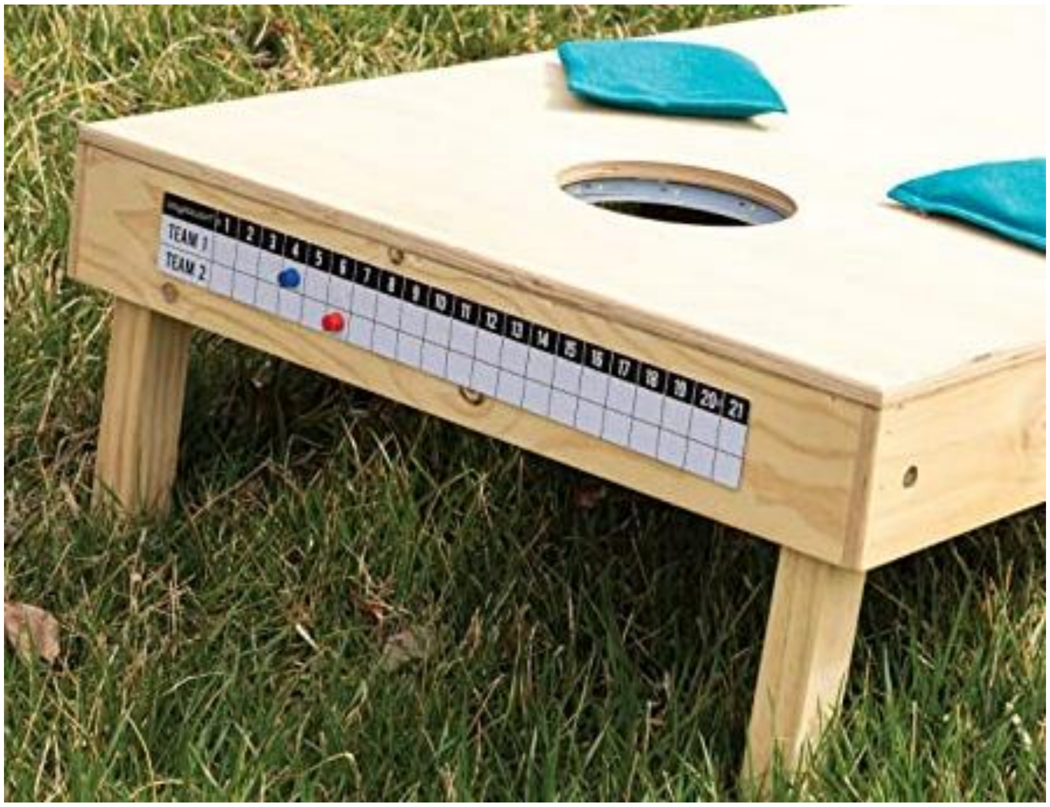


Figure 1.4: Manual Cornhole Scorekeeper [4]

1.3.2. Remote Scoring

A similar method to 1.3.1 is using a remote to keep track of the score. This involves a remote control and an LED scoreboard. After each round, a player would need to calculate the score of each round and input that number into the remote which will then show up on the scoreboard. Although this takes the stress of remembering the game's score, the player still needs to understand the game's scoring rules and input the correct score. [2]

Figure 1.5: LED Remote Scoring [2]

1.3.3. Auto-Scoring with RFID

Some engineers have tried using radio frequency identification to make an auto-scoring cornhole board. This system would use special tags inside the bags. The reader on the board, or in the hole, would then read each bag's unique tag upon arrival to identify a scoring value.

A team of electrical/computer engineers from the University of Central Florida made one working board as a proof of concept using ultra-high frequency radio identification (UHF RFID). They chose the ThingMagic Nano because it included UHF functionality and paired it with a Sparkfun Simultaneous M6E reader. They reported that this system had low power requirements, was fast to boot up, and could read 200 taps a second. Despite their success using the UHF

RFID, it was a significant portion of the project budget. Unfortunately, it seems like they were not able to make a fully functional device [4].

A California Polytechnic State University team used the same UHF setup with the ThingMagic M6E reader. However, they chose to use a linear vertical polarized antenna to get better coverage throughout the board. This choice came with the unintended consequence of only being able to read the tags in a vertical orientation. They solved this by placing two tags in each bag at 90 degrees to each other to always get one to read. It is unclear what kind of long-term durability this will have. This team came up with a clever way to attenuate the RF signal to identify when a bag was above or below the board. They made a diamond pattern under the boar with strips of aluminum foil to partially shield the signal. The team struggled with the Bluetooth communication system that was chosen for the project and although they had a working system, there was much room for improvement [4].

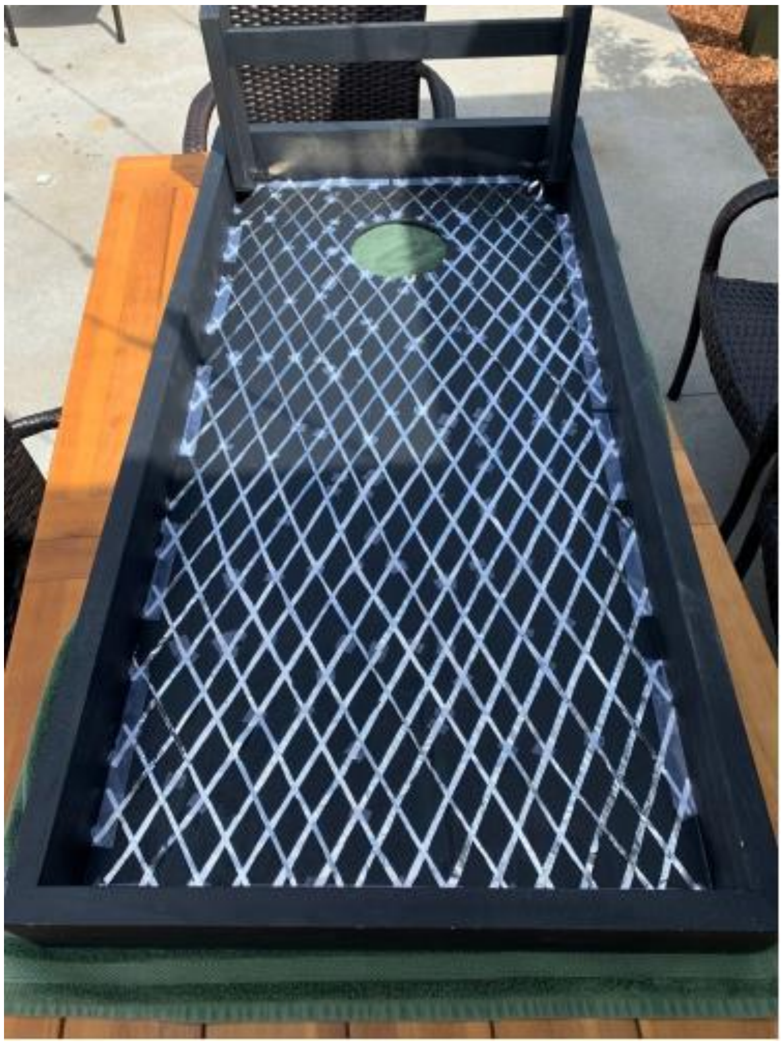


Figure 1.6: RFID Scoring [3]

1.3.4. Autoscoring with Image Recognition

A group of senior electrical engineers used a combination of a visual system and a color sensor to detect which bags have landed on the board or in the hole. They successfully got the camera to detect the separate bags and add both to the score. They also got the color sensor and camera to “communicate” with each other so that when a bag goes in the hole, only the color sensor will detect it and add it to the score. They did have some issues with the camera not being able to read 2 separate bags of the same color if they were on top of each other as well as the color sensor not being able to detect multiple bags going through the hole at the same time. The model was only half the true size of a cornhole board. Although the group had a semi-working prototype, there were several spots left for improving their model [5].

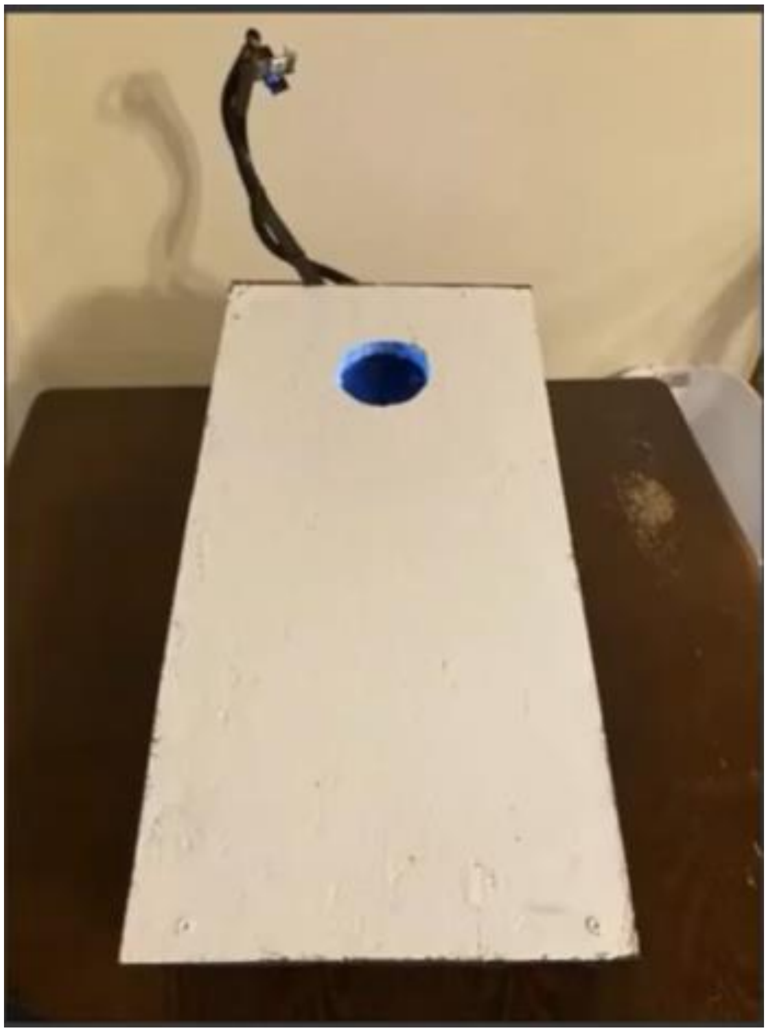


Figure 1.7: Image Recognition Scoring [4]

2. Objective Statement

The previous attempts either had flaws in functionality and/or aesthetics. There was also no way for the professional player to gain much from their boards. The objective of this project is to make a fully automated cornhole scoring system that is fully functioning, portable, has bag data extraction, and is aesthetically pleasing.

3. Specific Aims

For the auto-scoring system, a vectoring of pressure sensors is going to be used to detect how many bags are on the board and where most of that pressure and weight is distributed. This can also be used to detect where the bag landed for the bag data that can be extracted. The vectoring will be paired with a color sensor matrix to determine which colored bags landed on the board. The color sensor matrix will consist of multiple squares of color sensors to identify bag color at a certain location on the board. For the bags that go into the hole, a multi-sensor design will be implemented to ensure that all bags in the hole will be detected and recorded.

In the time where electronic games are more popular, this design helps preserve the classic game of cornhole with a technological appeal. It also brings people together in relax social setting. These are important cultural and social features for the design factor considerations located in Appendix B.

4. Requirements/Constraints/etc.

Some restrictions that the system must pertain to is the regulation size of the cornhole board. The board is a 4’ by 2’ board with a hole near the upper part of the board. The layout can be seen in Figure 4.1.

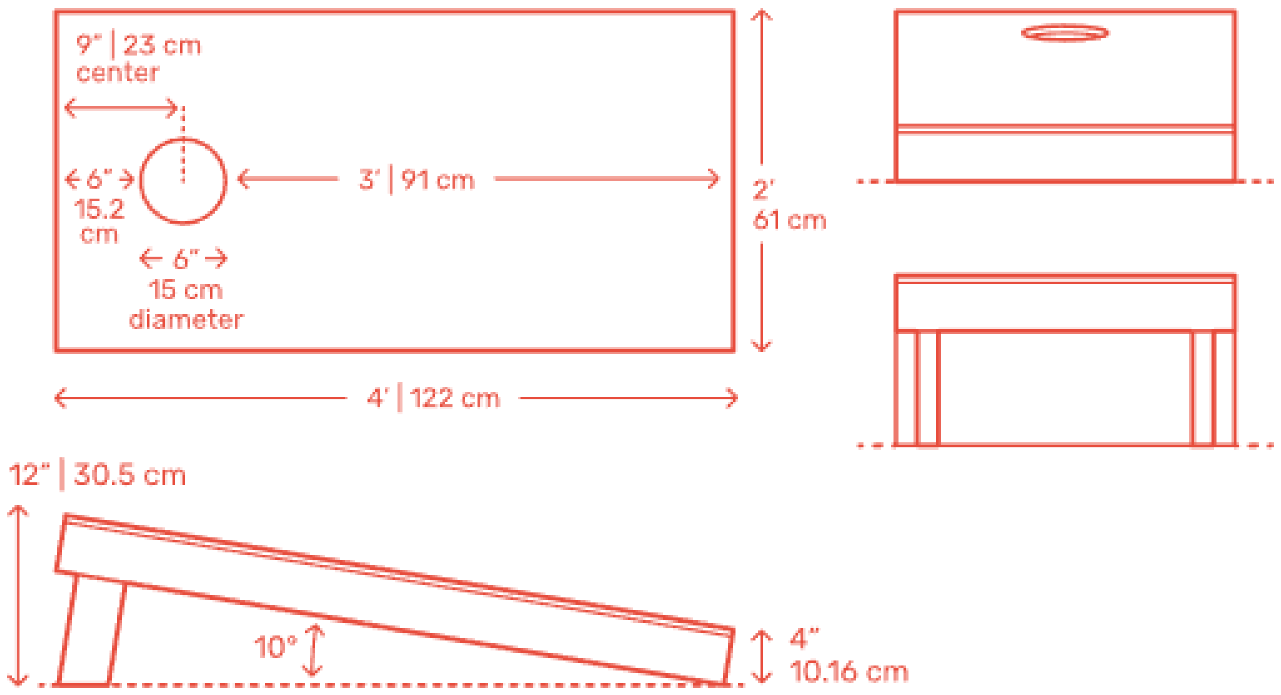


Figure 4.1: Universal Cornhole Board Dimensions [5]

The ability for the board to be stored away is also a huge concern for the average player. This means the board would need to light enough weight to move around as well as being able to break the board down enough to where it is a feasible job to store it away. This system should also work off DC power. This means that the board will not have to be plugged into an AC outlet while playing.

5. Code Logic

This section will go through an overview of what the boards and sensors will do with different scenarios. Team 1 is the red bags and Team 2 is the blue bags. This first scenario (Figure 5.1) is when Team 1 throws a bag in the middle of the board. The pressure sensors (highlighted in orange) will sense a change in equilibrium from their initial zeroed state. Then with the values that are generated from the pressure sensors, a general area of color sensors will be activated to read the color of the bag in that spot. Then when the bag is at rest, Team 1 will press their relative button on the back of the board which will then initiate the end of their turn and it will increase the current score of Team 1 by 1.

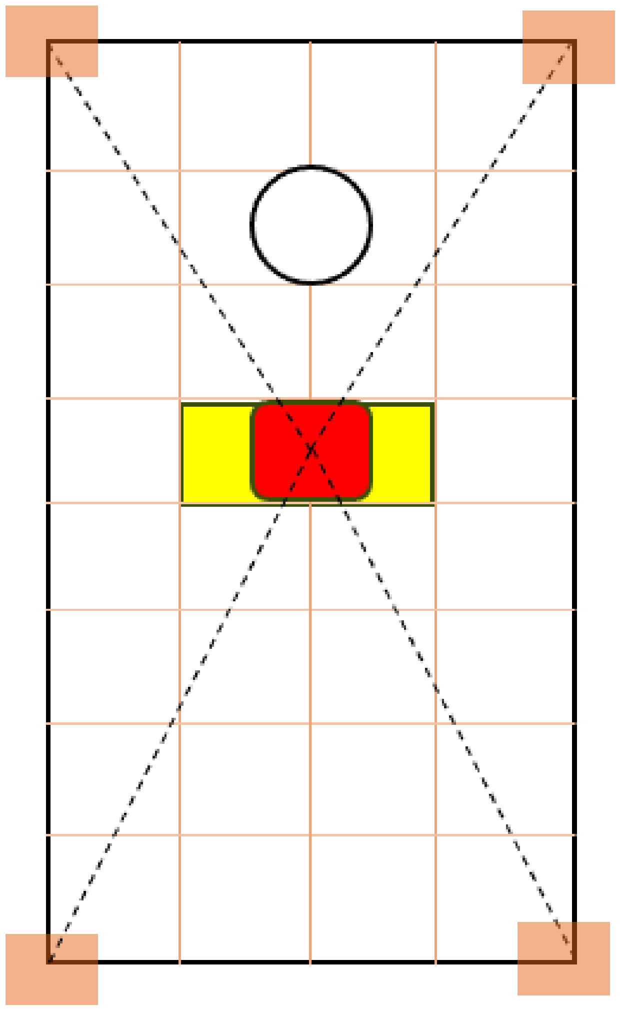


Figure 5.1: Team 1 Bag on Board

This second scenario (Figure 5.2) is when Team 2 throws a bag towards the bottom of the board. The pressure sensors will sense a change in equilibrium compared to the first bag. Then with the values that are generated from the pressure sensors, a general area of color sensors will be activated to read the color of the bag in that spot. Then when the bag is at rest, Team 2 will press their relative button on the back of the board which will then initiate the end of their turn and it will increase the current score of Team 2 by 1.

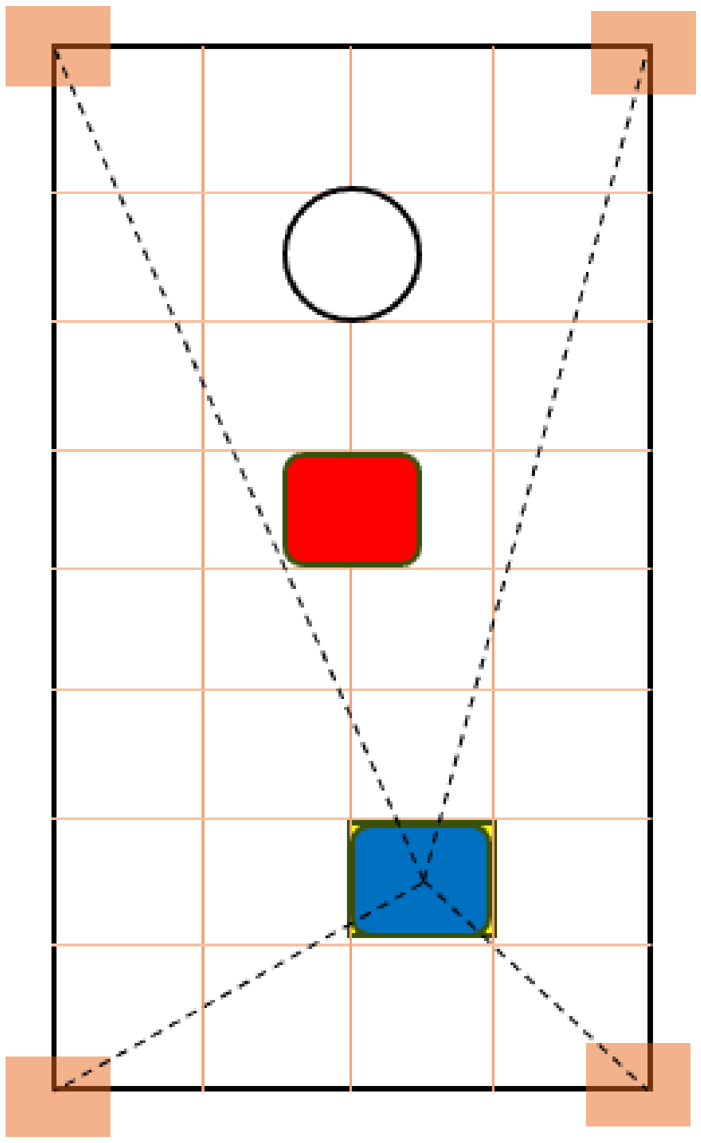


Figure 5.2: Team 2 Bag on Board

This third scenario (Figure 5.3) is when Team 1 throws a bag in the hole. Multiple sensors that are in the hole will detect that a bag has fallen in the hole and will read what color it is. Team 1 will press their button and the data will be successfully read and the bag will be released. Team 1's current score will increase by 3.

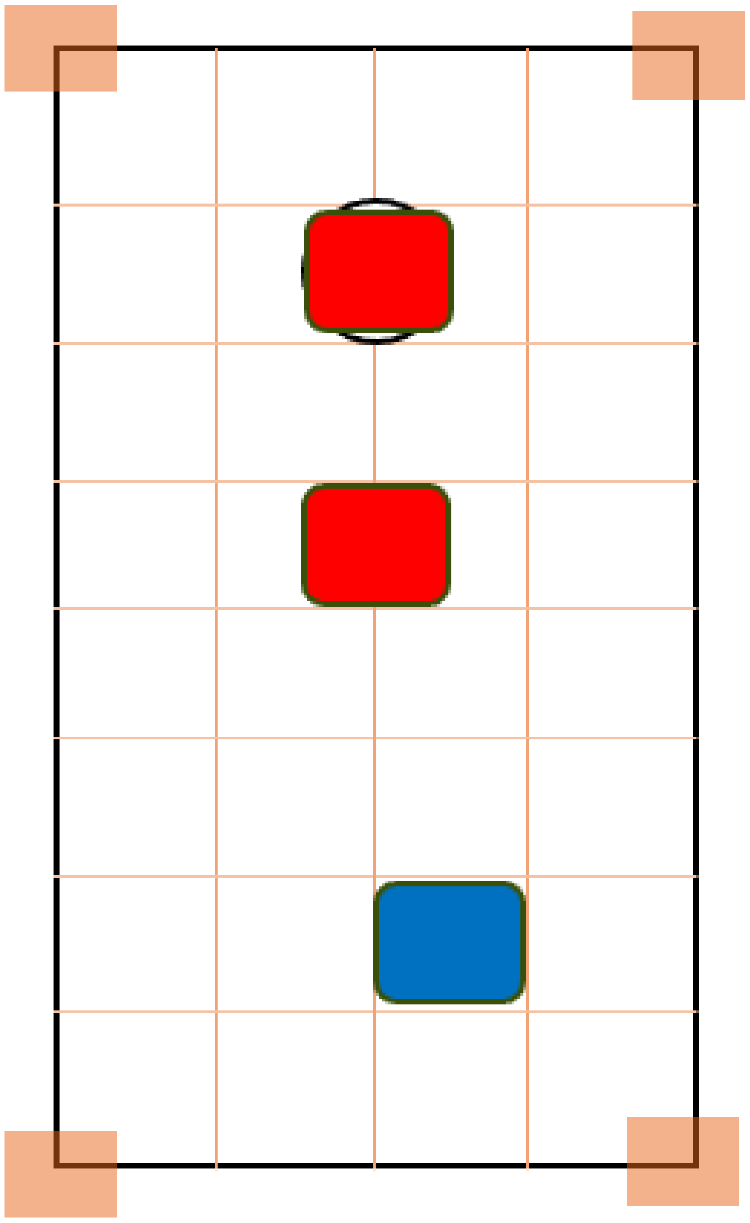


Figure 5.3: Team 1 Bag in Hole

This fourth scenario (Figure 5.4) is when Team 2 throws a bag, and it misses the board completely. Team 2 will press their button and when the readings are compiled, there will be no difference from the previous turn. This tells the board that a bag was missed and will initiate the end of Team 2’s turn.

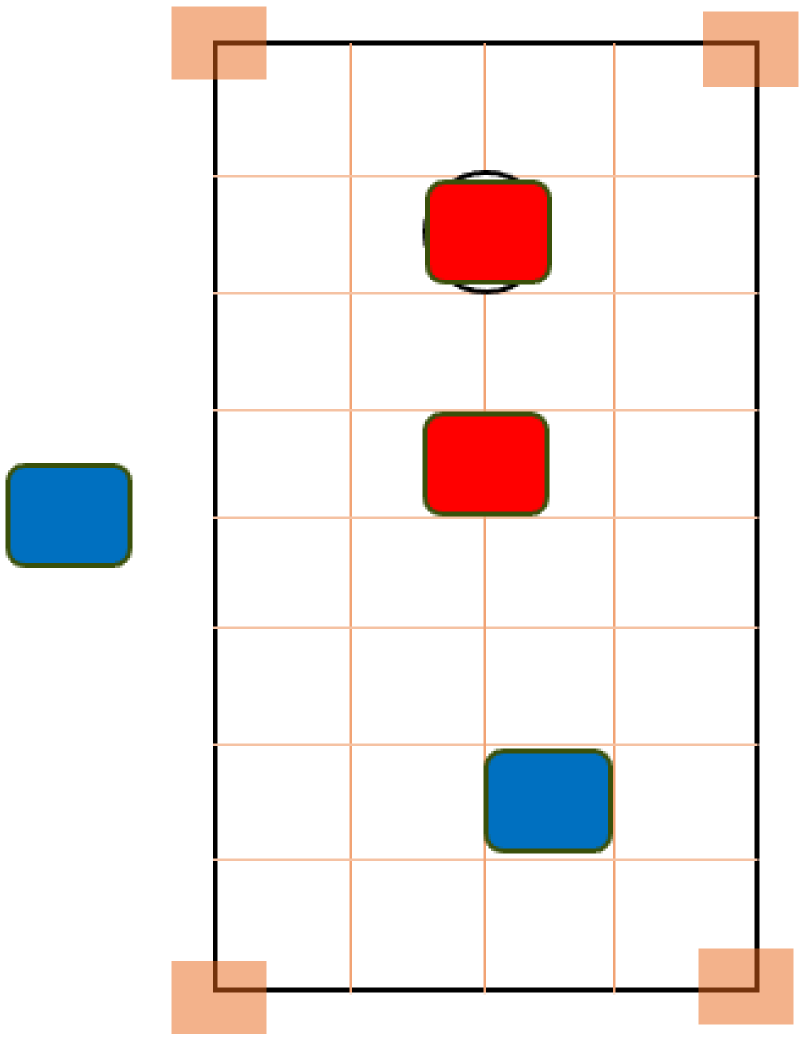


Figure 5.4: Team 2 Bag Misses

At the end of each round, the current score of each team will be compared to each other. If Team 1 has a higher current score than Team 2, the difference between the two teams will be given to Team 1 and vice versa for Team 2. For example, if Team 1 scored 4 points in round 1 and Team 2 scored 1 point in round 1, Team 1 has more points. This means Team 1 would have equated to 3 points and Team 2 would have 0 points after round 1.

This code logic continues until a team reaches a total of 21 points, in which the rounds and points will reset.

6. Project Design

This section goes into detail of a first idea of the cornhole board as well as all of the subsystems that are going into the design. The subsystems are explained to the reason that they were chosen and why they fit the system as a whole.

6.1. Design Idea

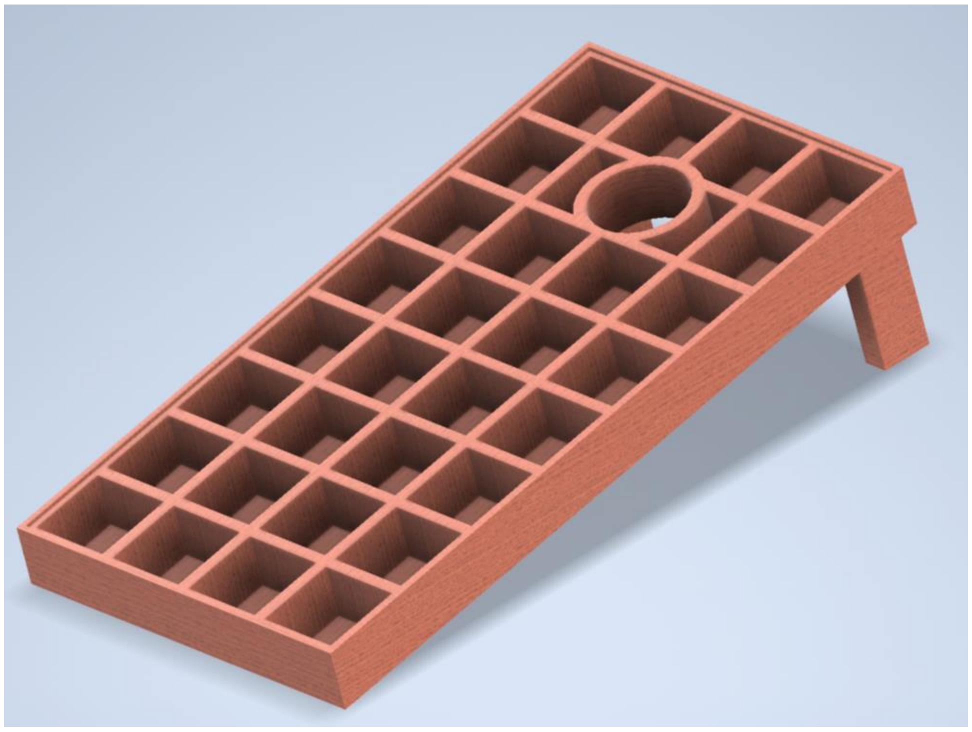


Figure 6.1: 3D Model of Design

Unlike the designs discussed previously, this design uses color sensors and pressure sensors to determine the bad color and location on the board and in the hole. The support frame will be made of 1”x4” wood planks, similar to a classic cornhole board. However, the top of this board will be made of clear plastic. Color sensors will be laid out in a matrix beneath the clear top to sense the color of the bags as shown in Figure 6.1. Pressure sensors will be mounted at each corner of the top to locate the position of the bags by measuring a change in pressure to each sensor. Color and pressure sensors will also be mounted in the hole to determine bags that have gone through. The legs of the cornhole board will be either removeable or foldable, so that the boards can be easily stored. The scoreboard will be at an angle so that the score can be seen without the players straining to view the score on the side of the board.

The block diagram in Figure 6.2 helps explain the gameplay logic. The color sensors communicate to the microcontroller (MCU) by the multiplexers and the pressure sensors communicate through the analog digital converter (ADC). Each bag that lands on the board or in the hole will be tracked and verified throughout the round of gameplay to identify knocking-off or shifting of the bags. When the round is over, the Board 1 MCU will update Scoreboard 1 and communicate the change in score to MCU 2 which will update Scoreboard 2. When round 2 is over, sensor data is processed and score updates are sent by MCU 2 to MCU 1.

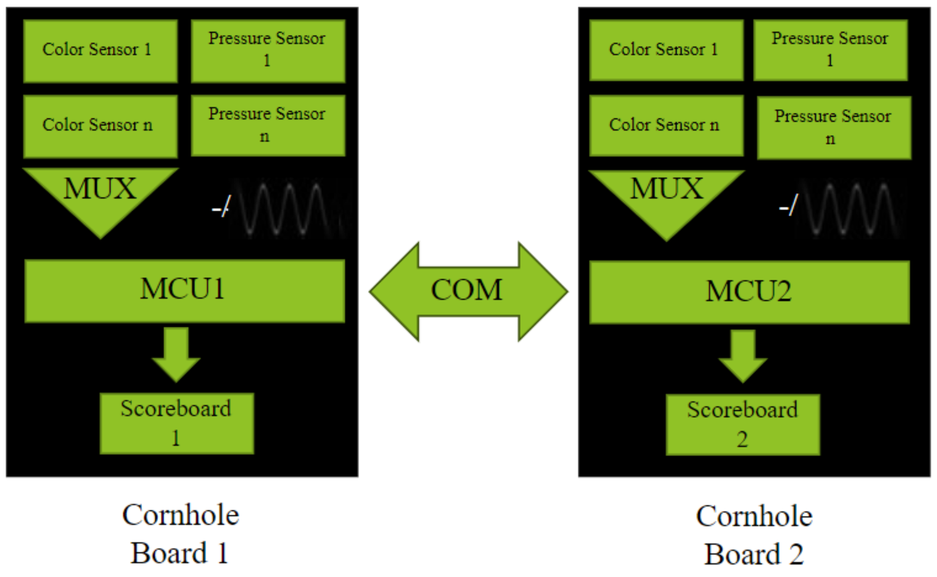


Figure 6.2: Gameplay Overview

6.2. Systems

The automated scoring cornhole system consists of a listed 5 subsystems: sensors, microcontroller, power, scoreboard, and physical board model. Each subsystem will be explained what the ideal purpose and contribution to the system. An explanation of which parts were chosen will be provided as well as considerations of other parts.

6.2.1. Sensors

The automated cornhole scoring system will use color and pressure sensors to determine the color of the bags as well as how many bags are on the board. The pressure sensor that was chosen was the SEN0296. This is a thin film style pressure sensor that has a range of 20g to 10kg, the widest range of all the sensors that were considered. The resistance also quickly drops to a linear behavior for ease of signal processing. This sensor will require signal processing. The SEN0294 was considered, but its range is smaller. The SF15-600 is similar in operation, but it is a long strip style that would not sense pressure directly at one corner.

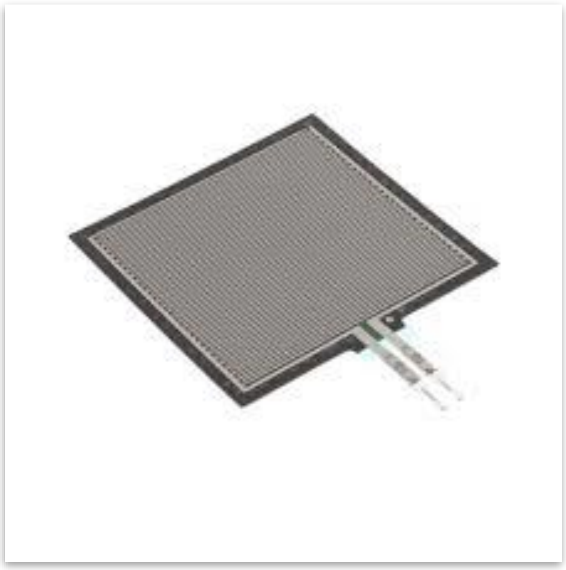


Figure 6.3: SEN0296 [7]

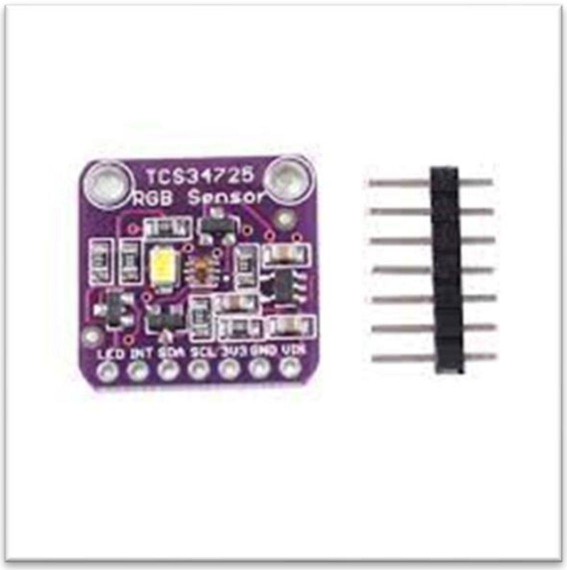


Figure 6.4: TCS34752 [2]

The color sensor chosen is the TCS34725. The reason this sensor was chosen over others is because of its I2C capabilities and no need for an analog to digital converter. However, it does require an I2C multiplexor. The two other color sensors that were in contention are the TCS3200 and the EACLSST3227A2. The TCS3200 has 4 on board LEDs that would illuminate the board top making the reading easier for the sensor, but it needs an analog to digital converter. The EACLSST3227A2 needs a custom PCB made for it. It could be made, but it would take more time as well as money to get it fabricated.

6.2.2. Microcontroller

The microcontroller chosen for the system is the ESP32. The reason this microcontroller was chosen over the Arduino microcontrollers is because it has Wi-Fi and Bluetooth modulation capability. It still has the ability to write in the Arduino IDE which will make it easy to get started. The pinout of the ESP32 can be seen in Figure 6.5. This MCU has I2C pins that will allow communication from our multiplexers.

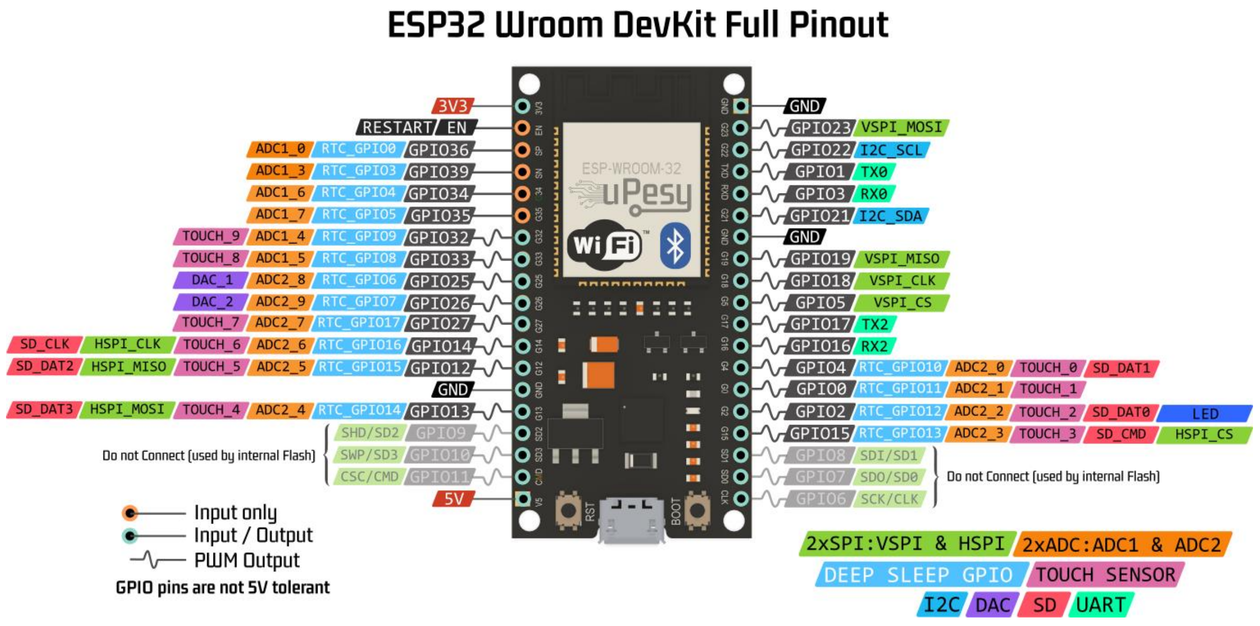


Figure 6.5: ESP32 Pinout [7]

The ESP32 can also be used to create a specific web application that can act as a Human Machine Interaction (HMI) between the customer and the cornhole boards. This addition could propel the project to another level of complexity. This web application could act as a scoreboard as well as a selector of the mode.

6.2.3. Power

The source of power was seen as one of two options: chargeable or rechargeable. A rechargeable battery pack seemed like the greater option because it wouldn't make the customer purchase regular batteries on a regular basis to ensure their cornhole boards have power. A rechargeable Lithium-Ion battery was chosen because it is one of the more common rechargeable batteries as well as long-living performance. The battery pack shown in Figure 6.6 uses a battery management system (BMS) that protects the cells from over charge, over discharge, and short circuit. In case of a short circuit, this could prevent fire. Also, the BMS used with rechargeable cells prolongs the life of the battery pack which results in less electronic waste. These are important health and safety as well as environmental features for the design factor considerations in Appendix B.

Figure 6.6: Lithium Ion Battery [3]

6.2.4. Scoreboard

A physical scoreboard will be included in the system. The score board is an e-ink display placed on the back side of the cornhole board. This display was chosen due to its low power mode and its outdoor readability. Since cornhole is mainly an outdoor game, it was preferred that these displays could be read in natural light and not tough for the players. The display only draws power whenever the display is updated, so if there is a delay in the game and the score isn't changed for a few minutes, the display would look like Figure 6.7 for those few minutes.

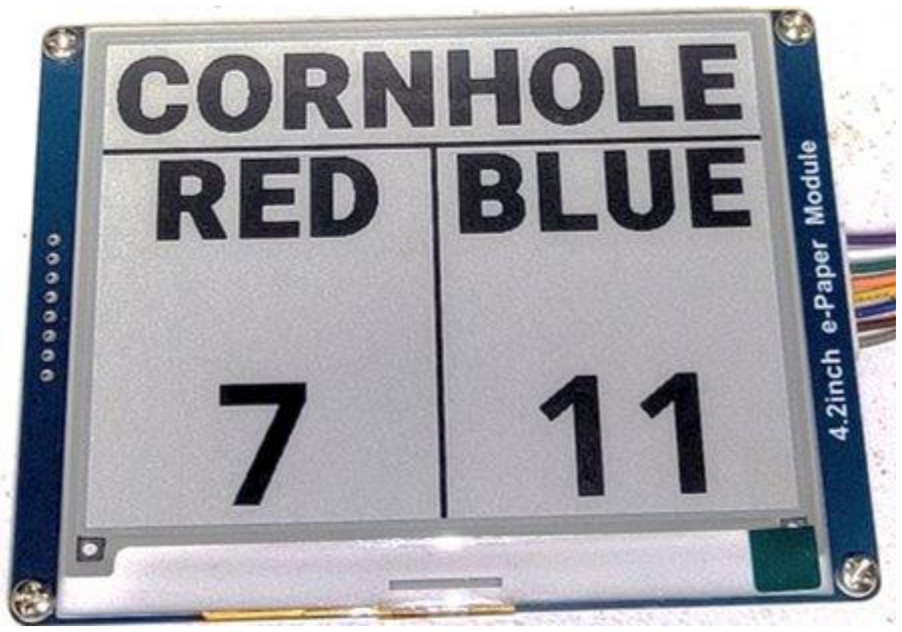


Figure 6.7: E-ink display

7. Game Board Design

The physical board was designed to be built using widely available materials so manufactures around the globe could build them. This means that this large product can be manufactured, assembled, and distributed locally anywhere in the world without the need for high shipping cost that come with large items. These are important global and economic design factor considerations located in Appendix B.

While designing the full-size board prototype, three versions were 3D modeled. Each time the board was redesigned to meet requirements that the previous board did not fulfill. Version 3 of the full-size prototype met all requirements, was built and assembled with all electrical components.

7.1. Early Designs

Before the final design, there were two versions that were updated and changed to better fit the project and give better results. Each section will explain the reason for each version, but then give the reasons why they were changed.

7.1.1. Version 1

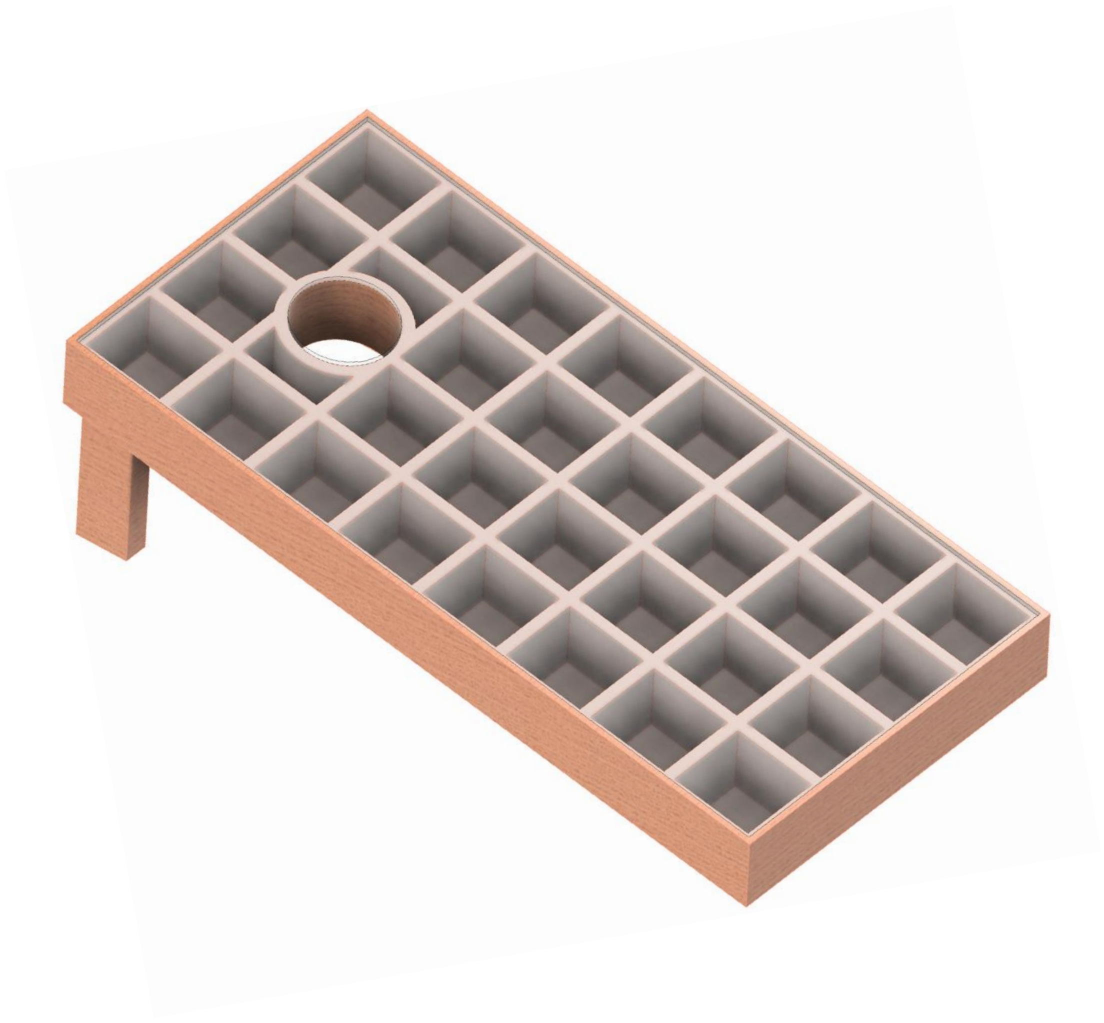


Figure 7.1: Board Model: Version 1

The first board was designed with the intent to implement pressure sensors at each corner and color sensors in a matrix below the board surface. A clear plexiglass top was included so the color sensors could read the bag colors. Color sensors would be mounted to the bottom of the recessed cubes shown in Figure 7.1. The walls of the grid patterns were designed to stop light coming in from the side from being read by the color sensor. The hole at the top was open like a normal cornhole board so bags could pass through. Pressure sensors were in the mounts for the feet.

This design had a few problems. The overall amount of wood used would make this design too heavy. Aside from being difficult to move, the weight also was too great for the pressure sensors that were selected. Additionally, the full height interior dividing walls did not allow for interior wiring.

7.1.2. Version 2

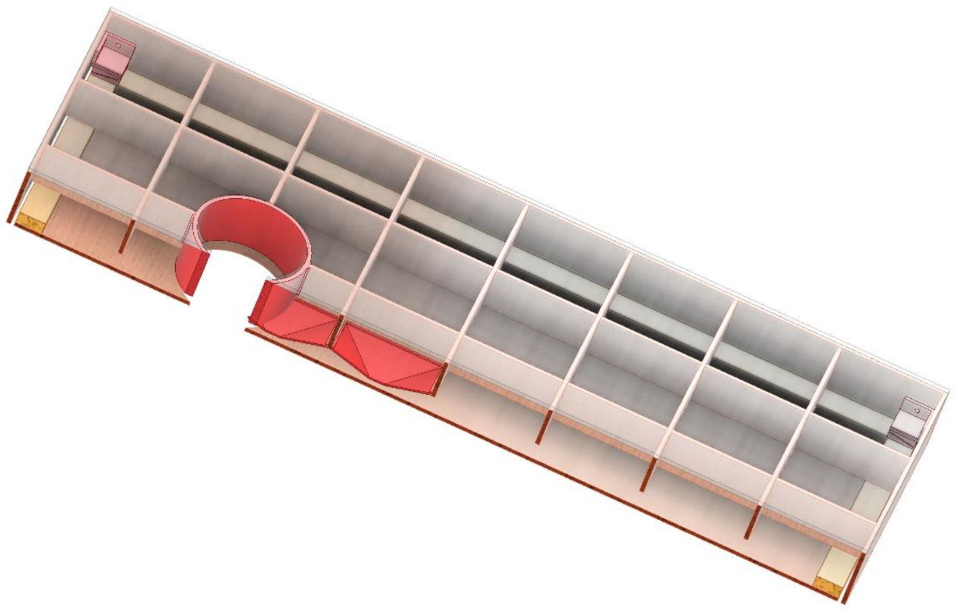


Figure 7.2: Board Model: Version 2 Half Section

The second board design had some dramatic changes to solve the problems from the first design. Figure 7.2 shows the version 2 board in a half section view to make the changes easier to see. First, the board is now split into two major sections, a framed bottom section, and a top section that includes the plexiglass surface and grid that is suspended above the frame leaving a small gap. The top section grid and outer panels are made from ¼ inch plywood to decrease weight while providing support for the plexiglass and shade the color sensors from light coming in from the side. In the corners, mounted to the top of the frame, are two-piece, 3D printed mounts that contain the pressure sensor between the two pieces and suspend the top above the bottom. The hole is also a 3D printed part that is incorporated into the grid walls and suspended above the bottom. Version 2 also included covers that would attach to the bottom of the grid and have a hole in the bottom to allow the color sensor to read the surface. The covers also provided a way to hide the wiring. The bottom section included a ¼ inch plywood panel to mount color sensors, electronic components, and hold wiring. The outer edges were framed with 1x2 inch wooden members to stiffen the panel and provide a strong mounting point for the feet.

Although this was a big step forward for the final design, there were some problems with this design as well. The mounts were too high and had the potential to block light from the corners of the board. The mounts also had no way to locate the top so that it set directly above the bottom. Version 2 also had no way to read bags that went into the hole. The bottom frame of this version was built to test rigidity, but the design allowed the bottom to twist too much which would have caused the pressure sensors to not read correctly.

7.2. Current Board Design

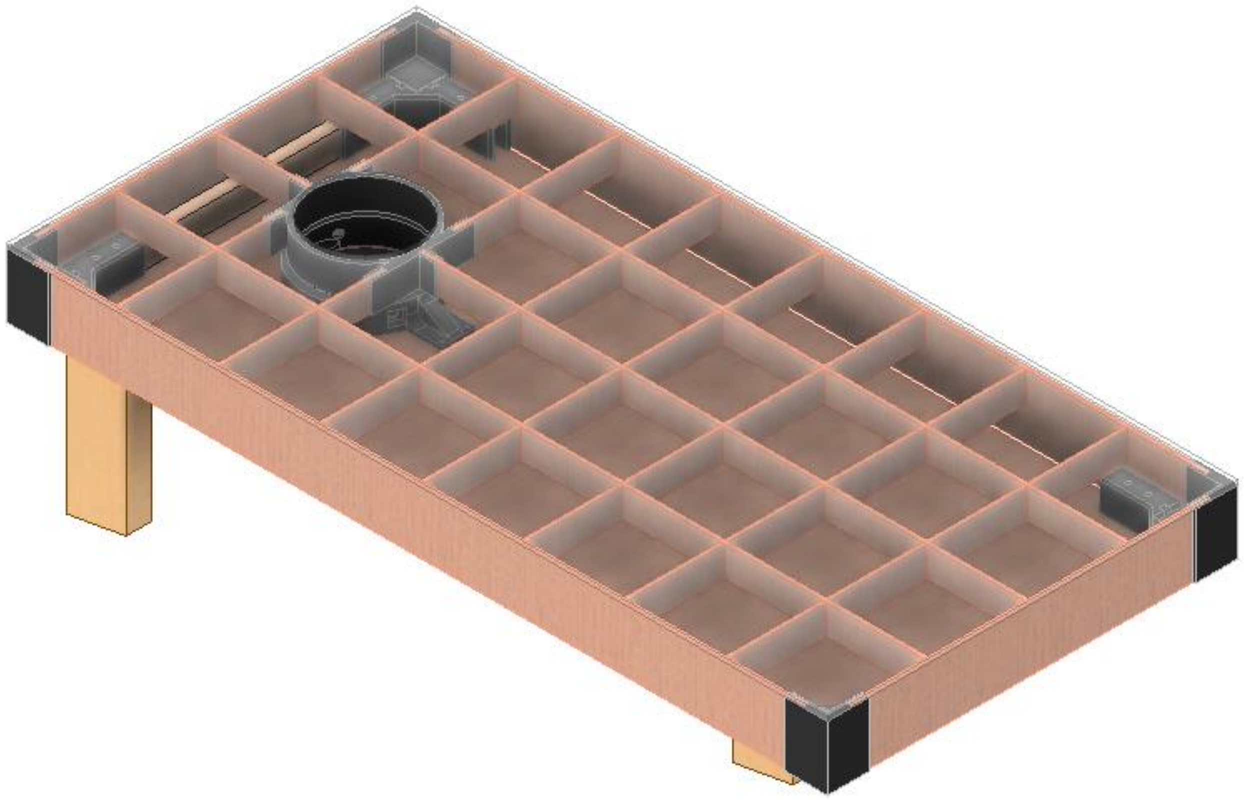


Figure 7.3: Board Model: Version 3

Version 3 of the full-size prototype looks similar to Version 2, but it has some significant changes. It still uses the two-piece, top and bottom design. First, to increase the rigidity of the bottom, the frame members were turned 90 degrees, so the wider part of the board was perpendicular to the bottom panel. This increased the bending moment so the frame would resist twisting. At the corners of the bottom frame, 3D printed brackets were designed to bolt to the frame members. These brackets were also designed with a recessed area to hold the pressure sensors. This design lowers the height of the sensor so the mounts block as little light as possible. The bracket recess also helps locate the top section so that it aligns uniformly on top of the bottom section. The bottom of the pressure sensor recess is at a 10-degree angle with the board surface so that the sensors are parallel to the ground. This ensures the entire weight of the bag is read between all four sensors.

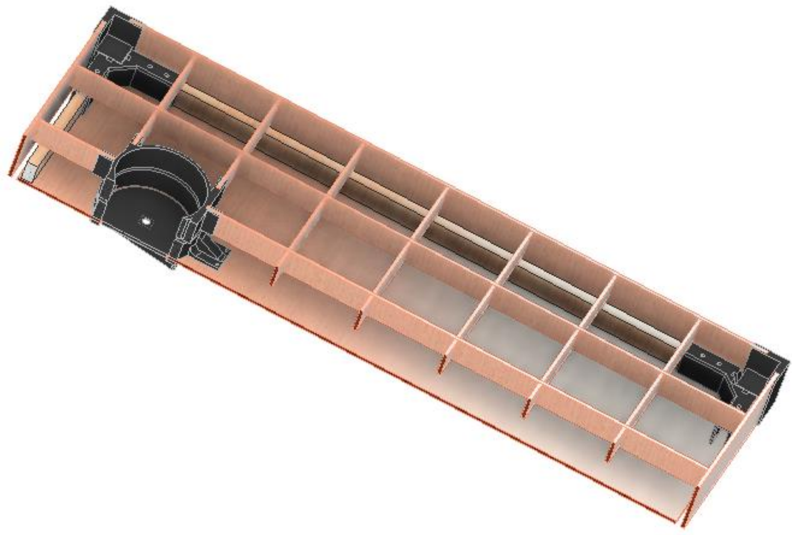


Figure 7.4: Board Model: Version 3 Half Section

The top section grid height was shortened to accommodate the taller members. The outer corners were designed to be 3D printed to incorporate the top part of the pressure sensor mount. The 3D printed top section hole piece was shortened to the same height as the grid to make room for a new hole design.

7.3. Hole Design

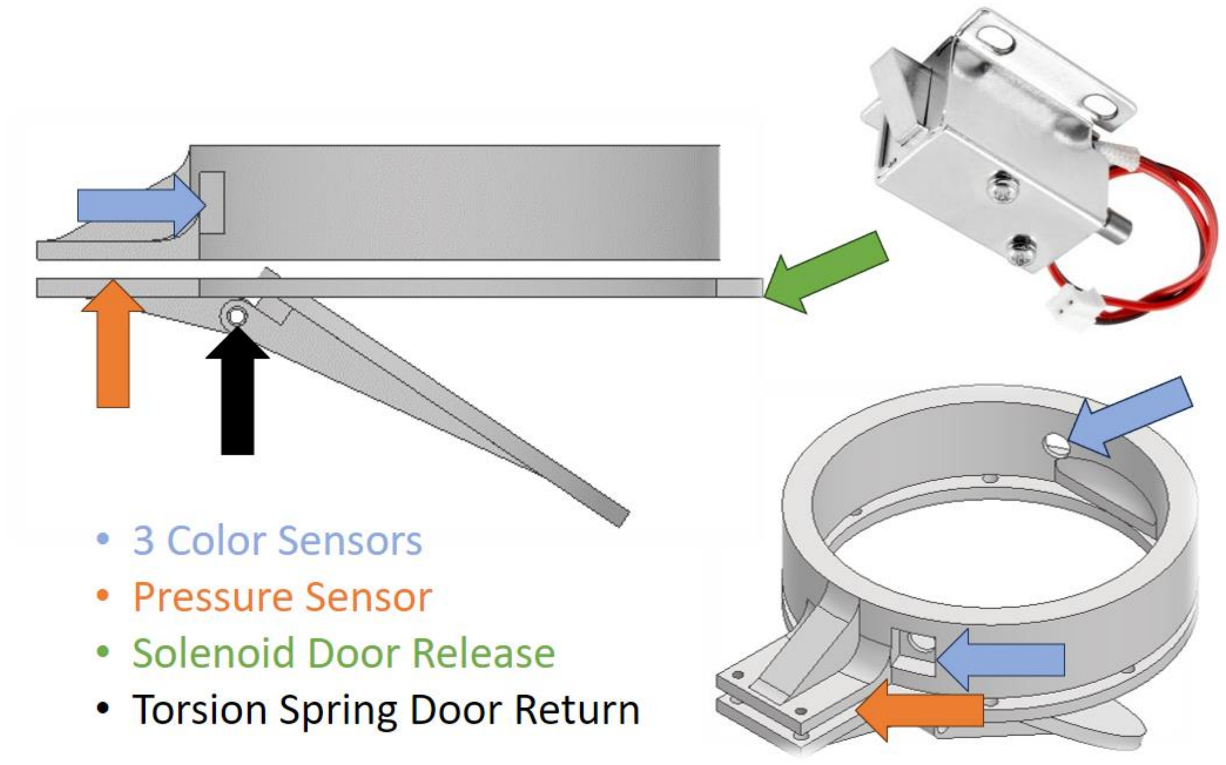


Figure 7.5: Hole Design

The previous board designs, versions 1 and 2, did not have a way to detect bags that went into the hole. However, version 3 allowed enough room to design a system to solve this problem. This hole design has three main 3D printed parts. The top part of the hole assembly mounts to the top of the bottom panel of the board. It has two places to mount color sensors on opposite sides. It also has a place to mount a pressure sensor. The bottom part of the hole assembly mounts to the bottom of the bottom panel of the board. ¼ inch bolts pass through the bottom of the hole assembly and through the bottom panel of the board and screw into treaded holes in the top part of the hole assembly. A hinged door is also mounted to the bottom part of the assembly. It has a torsion spring on the hinge so the door will return to a closed position when there is no bag on the door. The door is kept closed using a solenoid latch. The bottom of the door also has a color sensor mount.

Multiple color sensors are needed incase more than one bag goes into the hole at one time. The system is designed to detect when a bag goes into the hole, determine how many bags are present using the pressure sensor, detect the color of the bags, and then release the bags as soon as they are read. A detailed flow chart of this process is shown in Figure 7.6.

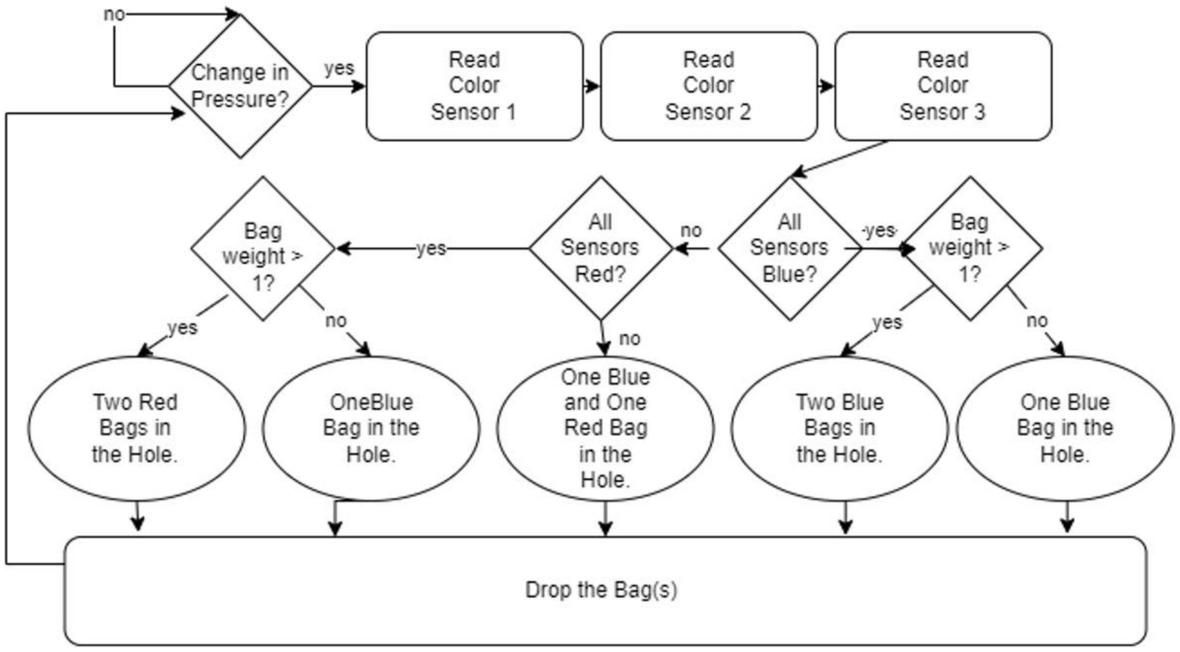


Figure 7.6: Hole Flow Logic

7.4. Whole Board Assembly

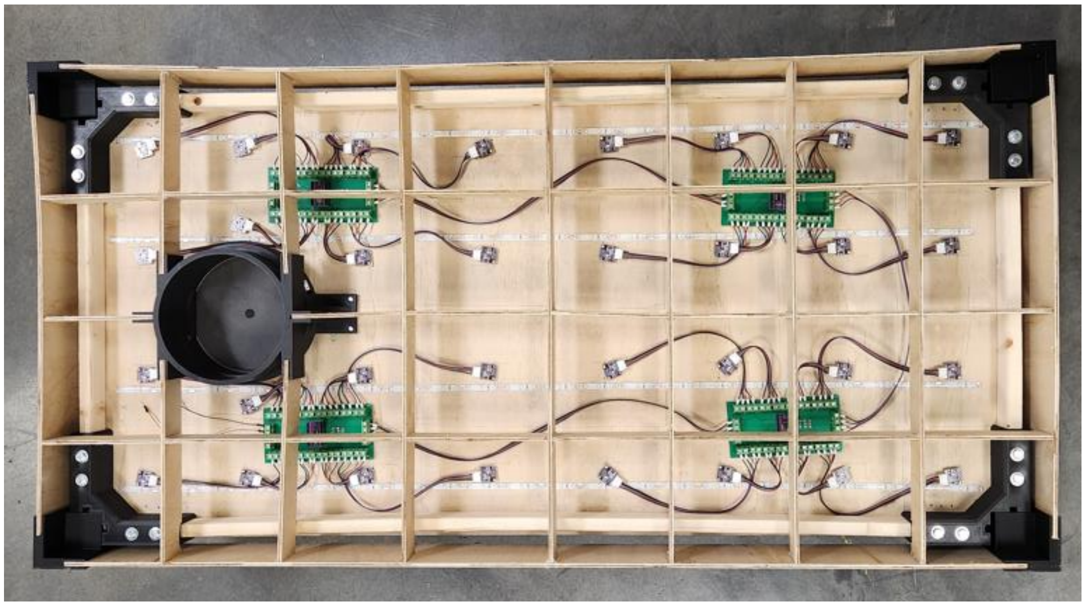


Figure 7.7: Physical Board Design

Figure 7.7 shows a picture of the assembled board with color sensors, multiplexer PCBs, and LED strips installed. The Plexiglass top has been removed for image clarity. This image shows the color sensors mounted in the center of each grid square. The shadows from the grid walls demonstrates that they do block some light from the side from reaching the color sensors.

8. Communication

Communication is a vital part of the project. This can be broken up into three categories: communication within the board, communication from the sensors to the microcontroller, and microcontroller to microcontroller (board to board). Each form of communication uses standard protocols developed for wireless or wired communication and each sub-section will go into detail about how these were accomplished.

8.1. Within the Board

Communication within the board includes actions within the code of the microcontroller that facilitate communication with each color and pressure sensor, as well as controlling backlights and the hole latch solenoid. General purpose input output (GPIO) pins of the ESP32 are used to turn on and off pressure sensor voltage dividers, backlights, and hole latch solenoid. Serial communication pins are used to control multiplexers to communicate with color sensors individually.

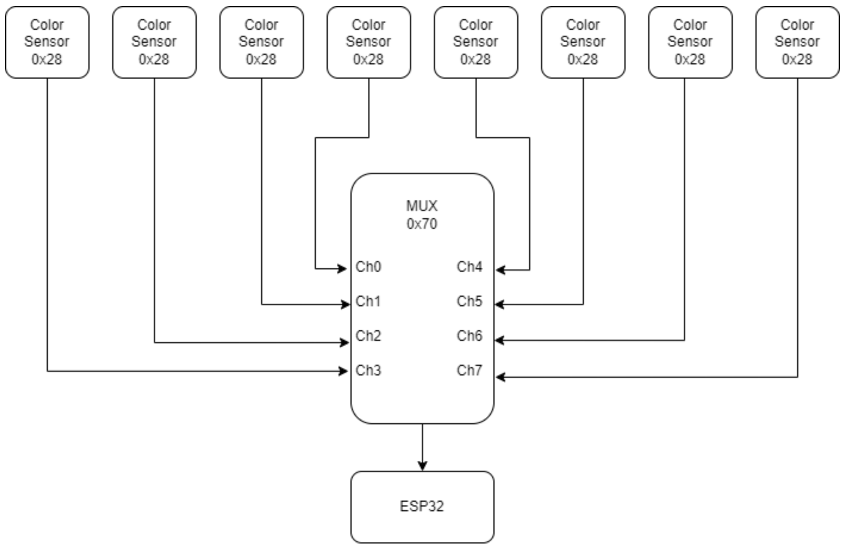


Figure 8.1: I2C Multiplexer Diagram

As mentioned in section 6.2.1, the color sensors use I2C communication to send information to the microcontroller. This is a serial communication protocol that uses two wires. When communicating using I2C, primary devices like microcontrollers communicate with a specific secondary device like a color sensor by sending the secondary device's address before issuing commands. However, the color sensors used for this board all have the same address and cannot be changed as shown in Figure 8.1. To communicate with an individual color sensor, a multiplexer is used. A multiplexer allows the I2C bus to be connected to one sensor at a time. The multiplexer used in this project is the TCA9548A shown in Figure 8.2. It has eight individual channels to which a color sensor can be connected. It also has three address pins that can be used to change the I2C address of each multiplexer. This is necessary because, since 35 color sensors were used, five multiplexers were needed.

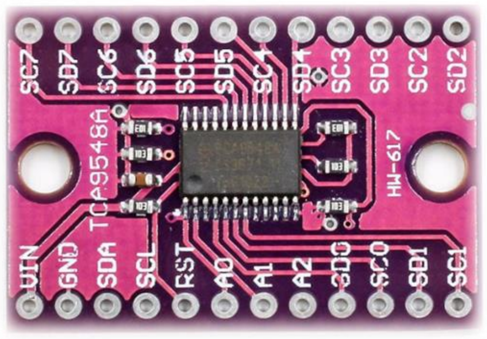


Figure 8.2: TCA9548A Multiplexer

To communicate with color sensors within the board, there are four main wiring lines that need to be ran through the board: positive voltage (V+). Ground (GND), data (SDA), and clock (SCL). The function of SDA and SCL will be talked about in detail in Section 8.2. To be able to distribute these lines effectively across a 4’x2’ board, a custom printed circuit board (PCB) was designed. This PCB can be seen in Figure 8.3. It also shows the connection of the screw terminals. The screw terminals following the first two pairs follow the same pattern going to the eighth pair.

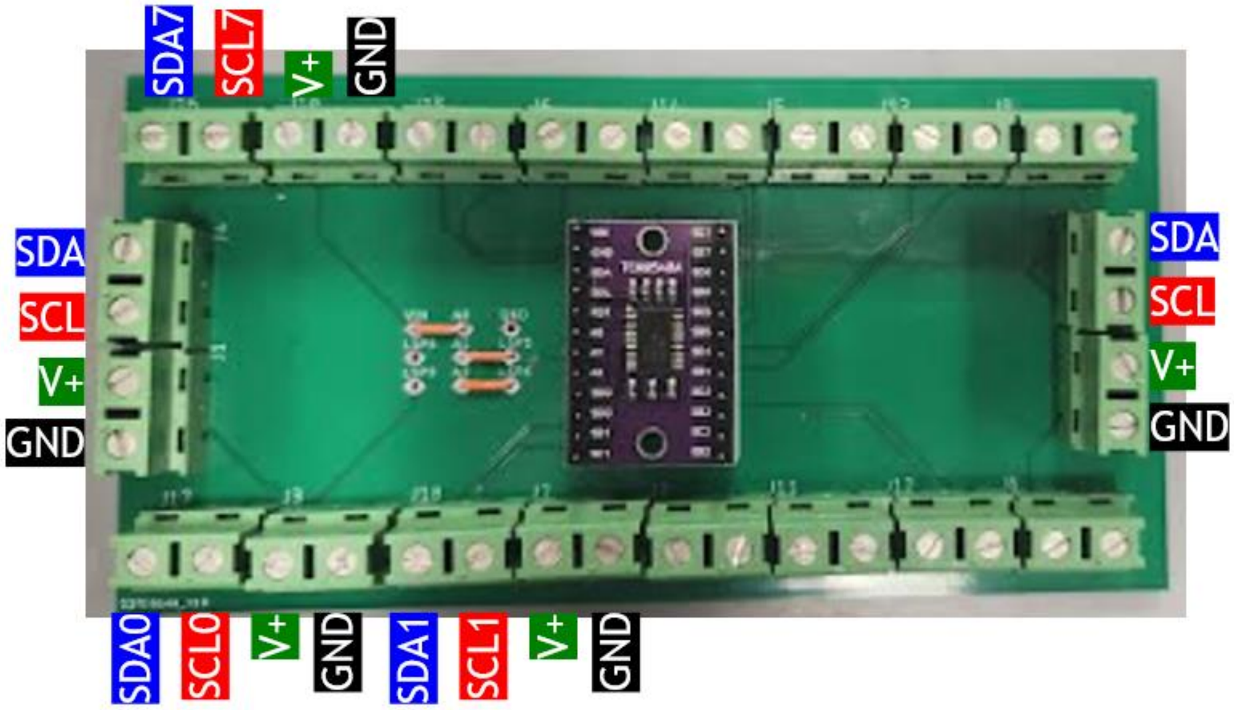


Figure 8.3: Multiplexer Printed Circuit Board

This PCB holds a multiplexer and 20 screw terminals. This allows each color sensor to be addressed individually as well as being able to distribute V+ and GND across the board without an immense amount of wire. This PCB is split into 2 main sections: color sensor connection and distribution. The color sensor connection sections and the top and bottom section of Figure 8.3. This allows the multiplexer to be connected to each color sensor individually. The distribution sections are the left and right sections of Figure 8.3. This allows the four main wiring lines to be distributed between the microcontroller and each PCB. Without the PCBs, each color sensor would need to connect to their specific multiplexer as well as a power source to be able to function. Figure 7.7 shows how these PCBs help diminish the amount of wire needed.

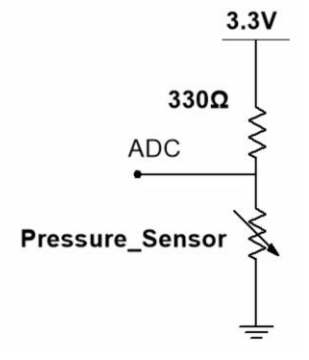


Figure 8.4: Pressure Sensor Voltage Divider

The pressure sensors act as variable resistors. They are placed into a voltage divider as shown in Figure 8.4. Since each of the five voltage dividers draw around 20mA, the 3.3V power source can be turned on and off as needed to reduce power consumption. The 3.3V power is provided by a GPIO pin on the ESP32.

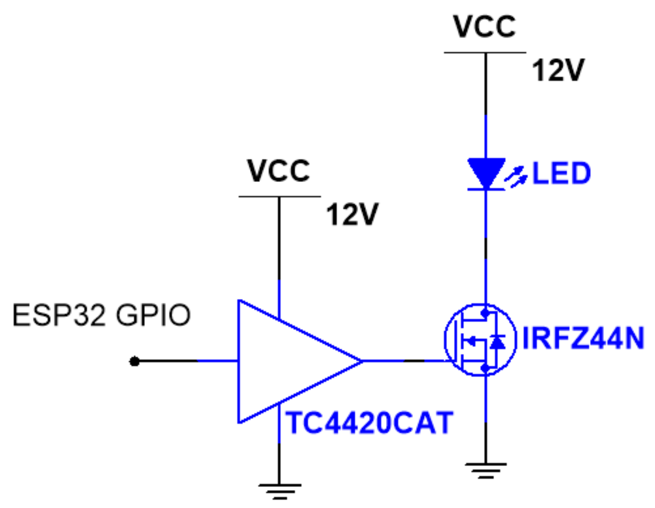


Figure 8.5: LED Control Circuit

The LED backlights and solenoid are also controlled using GPIO pins. However, both operate using 12V, so the 3.3V coming from the microcontroller cannot be used directly. An N-channel MOSFET was used to turn on and off the ground side of the 12V circuit for both devices as shown in Figure 8.5. The 3.3V signal from the ESP32 was also not a high enough voltage to fully turn on the MOSFET, so the TC4420 MOSFET driver IC was used to amplify this signal to 12V. The circuit to control the solenoid was the same as the LED control circuit.

8.2. Sensor to Microcontroller

Color sensors and pressure sensors require separate types of communication to communicate with the microcontroller. Since there are 35 color sensors (32 on the board and 3 in the hole), they need to be addressed individually, as stated in section 8.1, using multiplexers and the data and clock lines from the microcontroller. This is called Intra-Integrated Circuit communication or I2C communication. I2C uses two wires (Serial Clock and Serial Data) in a bus configuration as shown in Figure 8.6. To begin communication, the microcontroller will send a start bit to the serial bus to indicate to all devices that communication is starting. Then an address is sent to indicate with which device the microcontroller needs to communicate. Next, a single bit is sent to indicate whether this is a read or write command. Another single bit is sent to confirm. Information is then sent in a predetermined number of bytes and confirmed until the primary issues a stop bit.

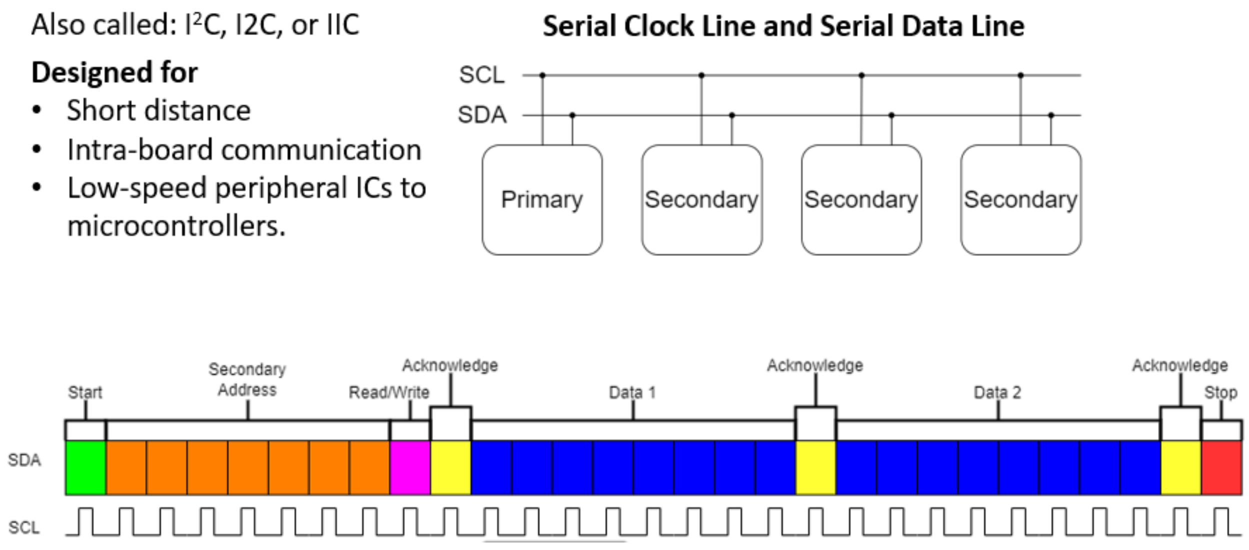
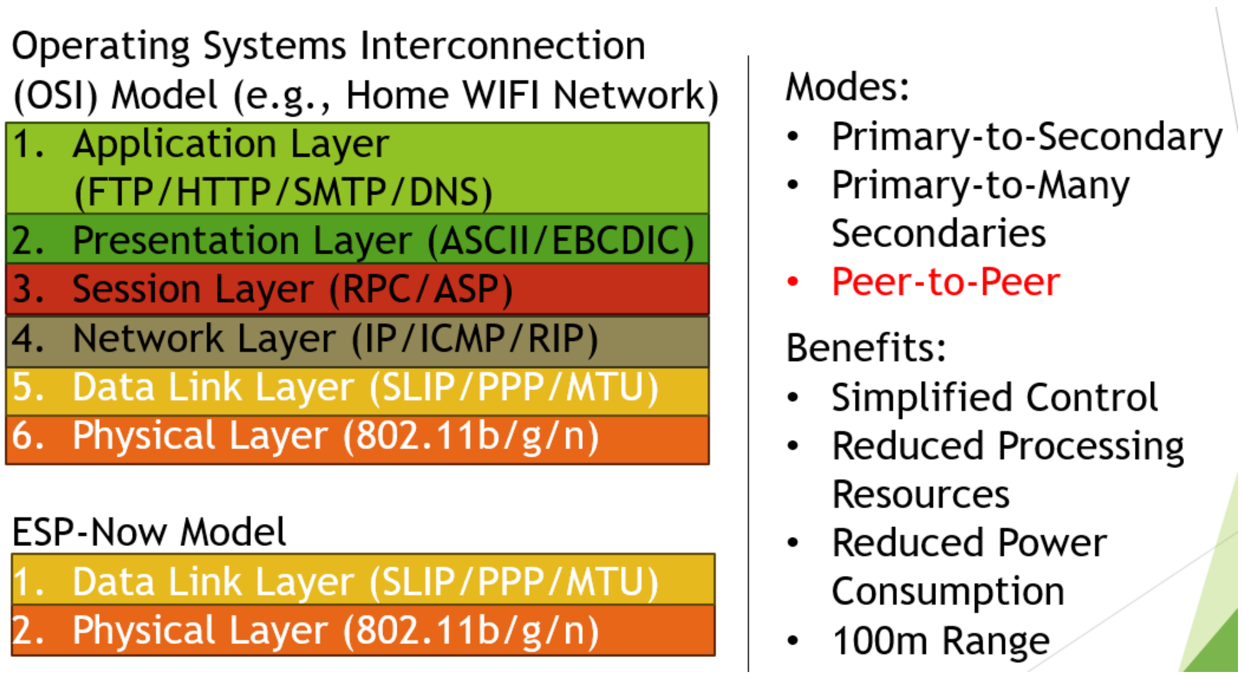


Figure 8.6: Intra-Integrated Circuit Communication

Reading the output of the pressure sensors is easier since serial communication is not involved. An analog to digital converter (ADC) pin in is connected to the middle of the voltage divider, as shown in Figure 8.4, to read the change in voltage across the pressure sensor. The ADC converts the voltage reading into a 12-bit value. This is how the microcontroller determines the pressure at each sensor.

8.3. Board to Board

It was established that wireless communication was ideal for this project. The ESP32 has a special protocol called ESP-NOW. ESP-NOW is a communication protocol developed by Espressif Systems, specifically for their ESP8266 and ESP32 Wi-Fi modules. It enables low-power, peer-to-peer communication between devices without the need for a traditional Wi-Fi network infrastructure. ESP-NOW is well-suited for IoT applications, providing a fast and efficient way for ESP8266/ESP32 devices to exchange data directly, making it ideal for scenarios where power consumption and quick data transmission are critical. As seen in Figure 8.7, the ESP-NOW protocol skips the top 4 layers of typical operating systems interconnections such as home Wi-Fi. This protocol sets up a “primary-secondary” architecture similar to the I2C communication which allows each ESP to send data to each other and use that data. The reason to have data sent from both ESPs to each other is to communicate the score between the boards, so the game can progress evenly and not have different values of the team's score. The primary ESP32's communication code can be found in Appendix G and the secondary ESP32's communication code can be found in Appendix H. This wireless communication uses the standards from the Federal Communication Commission section 15 as stated in Appendix B.



Operating Systems Interconnection (OSI) Model (e.g., Home WIFI Network)

1. Application Layer (FTP/HTTP/SMTP/DNS)
2. Presentation Layer (ASCII/EBCDIC)
3. Session Layer (RPC/ASP)
4. Network Layer (IP/ICMP/RIP)
5. Data Link Layer (SLIP/PPP/MTU)
6. Physical Layer (802.11b/g/n)

ESP-Now Model

1. Data Link Layer (SLIP/PPP/MTU)
2. Physical Layer (802.11b/g/n)

Modes:

- Primary-to-Secondary
- Primary-to-Many Secondaries
- Peer-to-Peer

Benefits:

- Simplified Control
- Reduced Processing Resources
- Reduced Power Consumption
- 100m Range

Figure 8.7: ESP-NOW Protocol

9. Testing

This project went through two small prototype setups before moving to a full-size prototype. Each of the prototypes were used for different testing applications. The first small-scale prototype was uses for testing different kinds of sensors. The second small-scale prototype was used to test microcontrollers and communication. The large-scale prototype was used to test all systems individually.

9.1. Prototype

The first prototype can be seen in Figure 9.1. This is a 1'x1' box that simulates a section of a cornhole board. The prototype uses 4 color sensors (one in each quadrant), 4 pressure sensors (one in each corner), 1 multiplexer, and 1 microcontroller. The microcontroller for the first prototype was an Arduino Mega because of the versatility of the IDE and all of the open resources that help with building a code. The Mega also has a lot of pins available allowing for a change in sensors if needed.

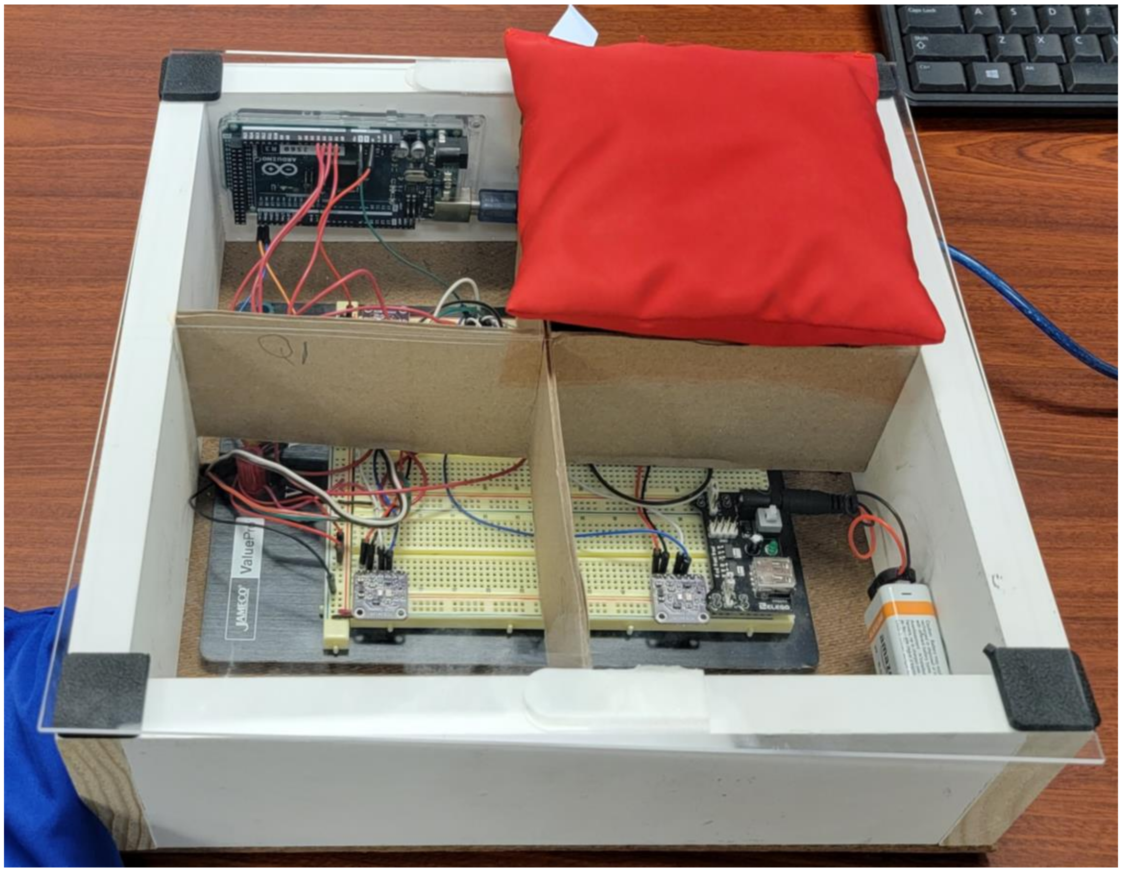


Figure 9.1: Prototype Sensing Red Cornhole Bag

For the second prototype, the Arduino Mega microcontroller was switched to an ESP32 for its wireless communication capabilities as well as still being able to use the Arduino IDE. A second multiplexer was also used to control two of the color sensors to test the control of the multiplexers.

Using the small prototype, the code in Appendix F was used to gain data for both the pressure sensors and the color sensors. Figure 9.2 showcases the digital value of the pressure sensors when there are no bags on the board.

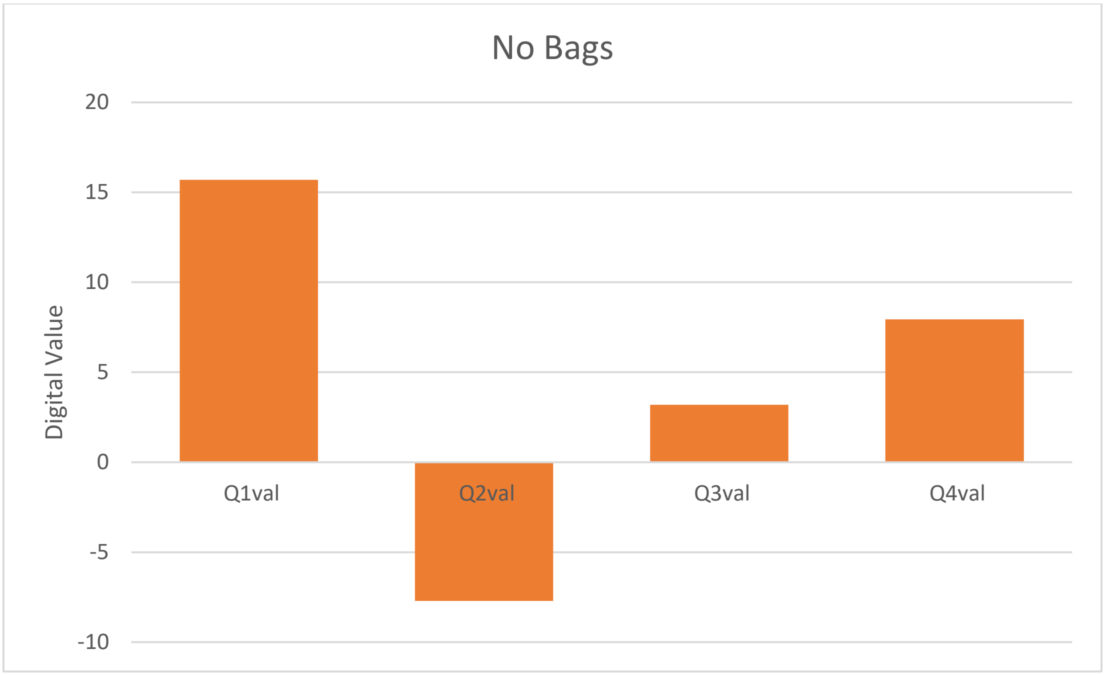


Figure 9.2: Pressure Sensor Values with No Bags on Board

Figure 9.3 showcases the digital values of when there is a bag located in Quadrant 1. All of the values jumped up, but Quadrant 1 jumped immensely compared to the rest.

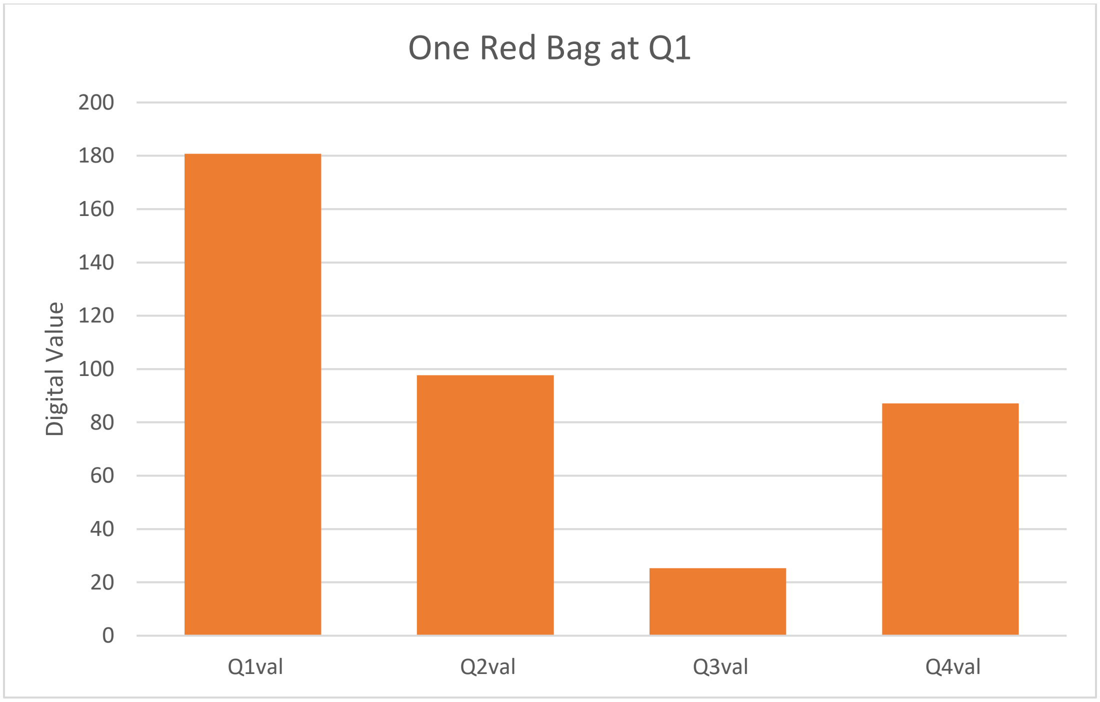


Figure 9.3: Pressure Sensor Values with Bag Located in Quadrant 1

Figure 9.4 showcases the digital values of when a bag is located in between Quadrant 1 and Quadrant 2. Compared to Figure 9.3, the values of the are quite different.

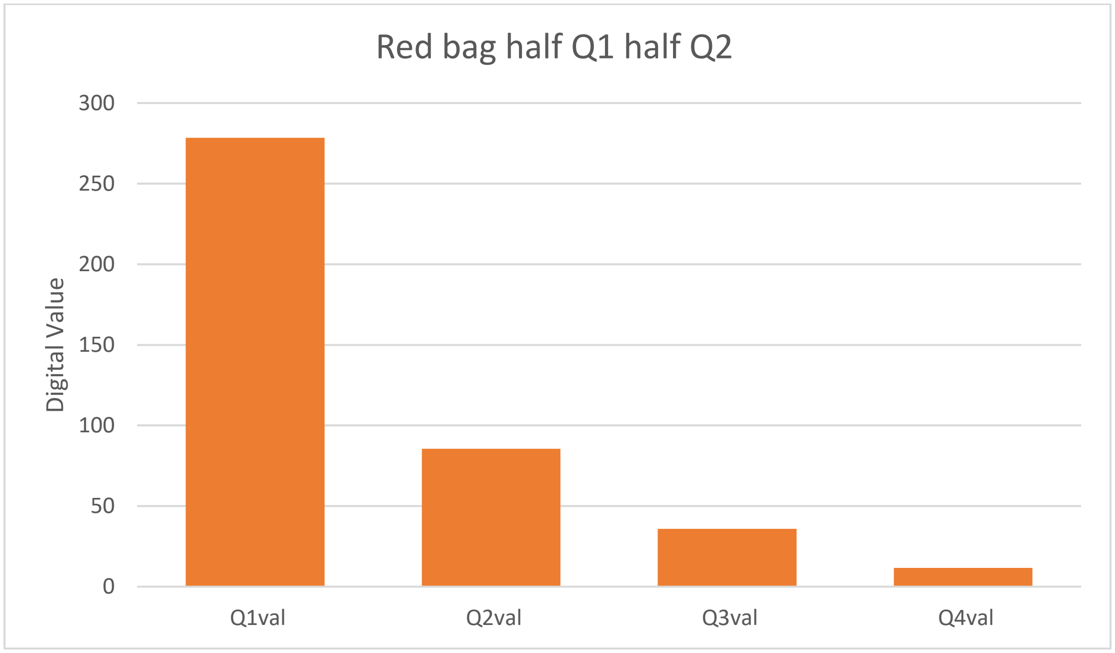


Figure 9.4: Pressure Sensor Values of Bag Located Between Quadrant 1 and Quadrant 2

Figure 9.5 showcases digital values of the color sensors when there is no bag on the board. All of the color values, besides blue, are at the maximum digital value.

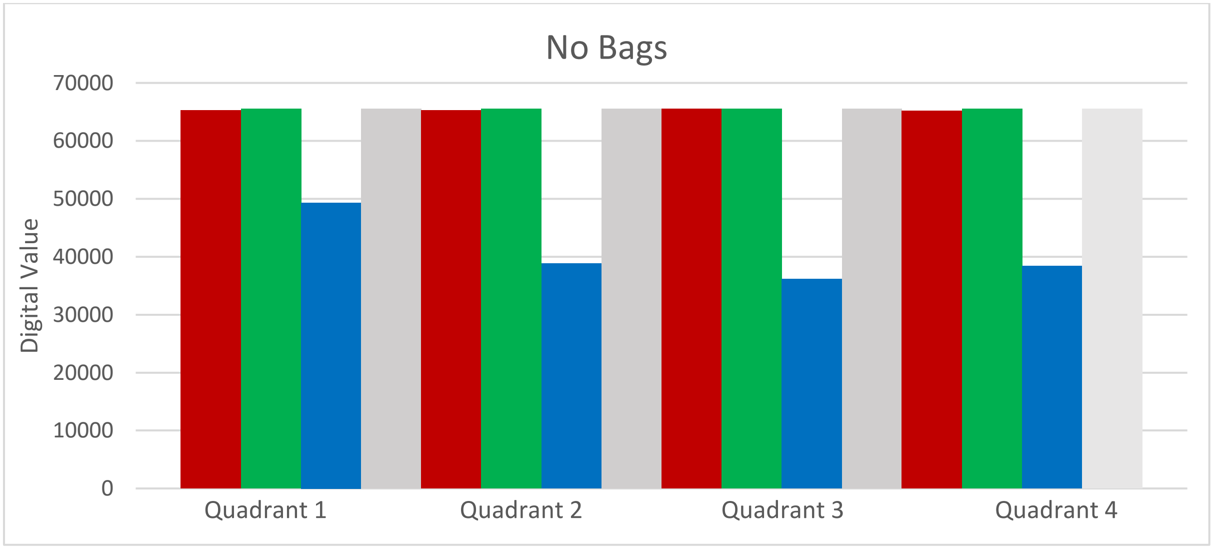


Figure 9.5: Color Sensor Values with No Bag on the Board

Once a bag lands on the board, that is when the values start to change. Figure 9.6 showcases digital values of the color sensors when there is a blue bag in Quadrant 1. It can be seen that all of the values in Quadrant 1 dip, besides the clear value. It takes a conditional to be able to identify that it is a blue bag that has been detected.

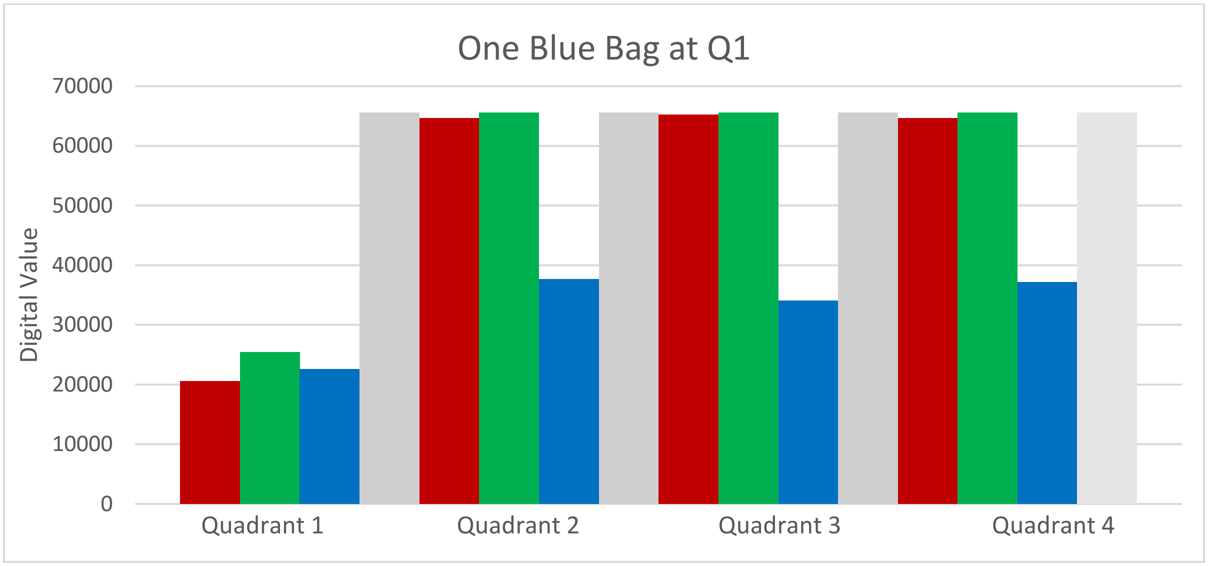


Figure 9.6: Color Sensor Values with Blue Bag Located in Quadrant 1

Figure 9.7 showcases digital values of the color sensors when there is a red bag in quadrant 1. This figure shows similarities to the previous one, but the red value sits at a much higher value than before. This allows the group to know that a red bag is being read.

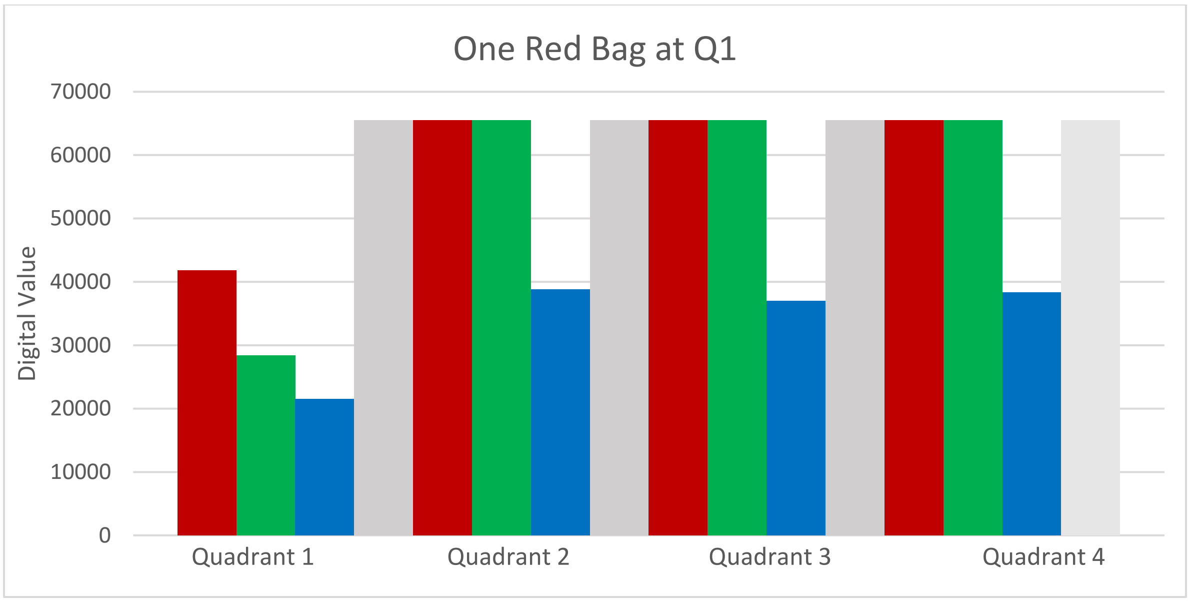


Figure 9.7: Color Sensor Values with Red Bag Located in Quadrant 1

Figure 9.8 showcases digital values of the color sensors when there is a red bag located in between quadrant 1 and quadrant 2. Both Quadrant 1 and Quadrant 2 have a change in digital value. Similarly, to Figure 9.7, the red values sit much higher than the rest of the values in their specific quadrant.

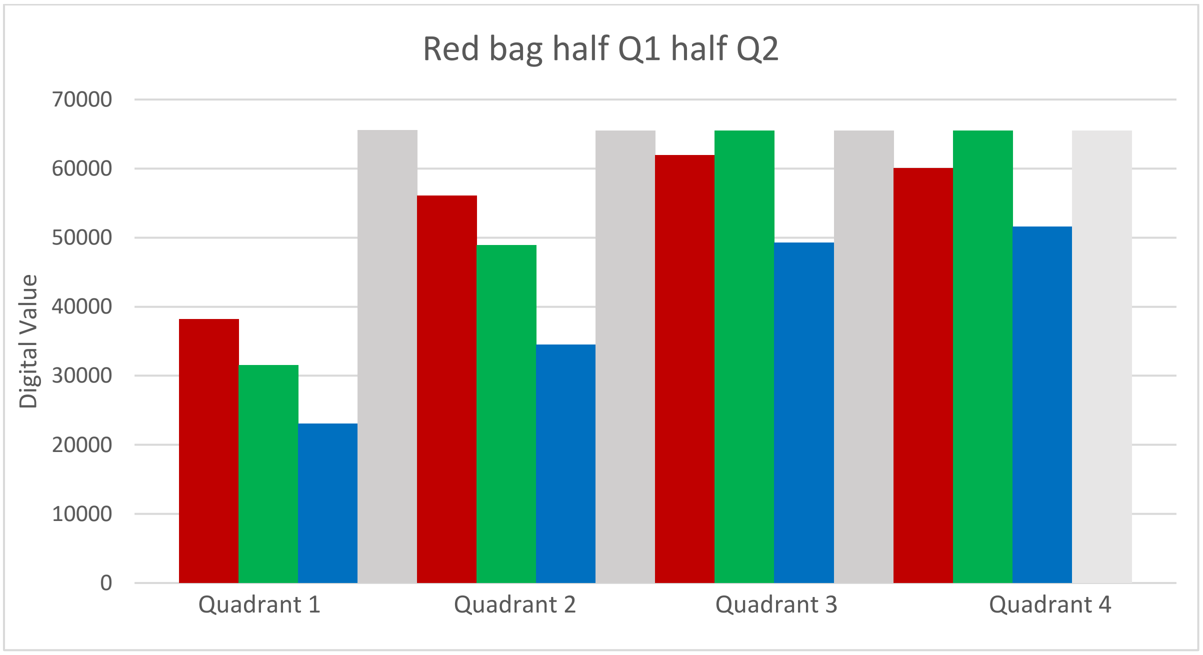


Figure 9.8: Color Sensor Values with Red Bag Between Quadrant 1 and Quadrant 2

These results furthered the group’s confidence in the project because these values were able to be separated and displayed. It also helped prove the concept of the project as a whole.

9.2. Testing Plan

After testing the prototype and getting successful reading from both sensors, expansion into the full-sized board was the next area to go. The first electrical test is being able to get all the components to run off a singular DC power supply. This leads to testing each quadrant of sensors, which consists of eight color sensors connected to one PCB and one pressure sensor. This test will consist of similar tests that were ran on the prototype to receive similar results.

Some individual tests that were tested are the ability to control both the backlighting and the solenoid. The backlighting needs to be controlled by a pulse-width modulation to allow it to brighten and dim depending on the brightness of the surrounding area. The solenoid needs to be controlled by an electrical signal from a specific GPIO pin, so that when a bag lands in the hole, the solenoid will retract and drop the bag through based on whether that GPIO pin is high or low. Apon assembly for the board components, each component system was tested using the code found in Appendix J through Appendix Q.

After the components can be controlled and have power delivered to them, the code is then verified. The code consists of an expanded version of the prototype code, the gameplay code, and the communication code. The prototype code consists of pressure sensor and color sensor readings. The gameplay code uses the readings from the previous code to determine when either team scores or not. The communication code then takes each team's final score of each round and communicates to the other ESP32 and updates that score.

10. Results

Applying the test plan stated in section 9.2, there were experiences of accomplishments as well as some areas that didn’t fully show what was trying to be shown. Section 10.1 shows the accomplishments that the group experienced, and section 10.2 shows some of the problems that were faced.

10.1. Accomplishments

This project did face some issues, but there was also a great deal of accomplishments. The first of them is being able to prove the concept of the project using the prototype. The prototype allowed us to show the values received from the color sensors and pressure sensors as shown in Figure 9.2 through 9.8. Since this project was started from scratch, the entire design process was needed. From thinking of an idea to designing it, then building it and testing it. It allowed the group members to experience a full process of what the real world could hold. The adjustable backlighting control was achieved. A PWM signal was sent to each LED strip and was brightened and dimmed. Figure 10.1 shows the full-sized board with its LED strips lit up.

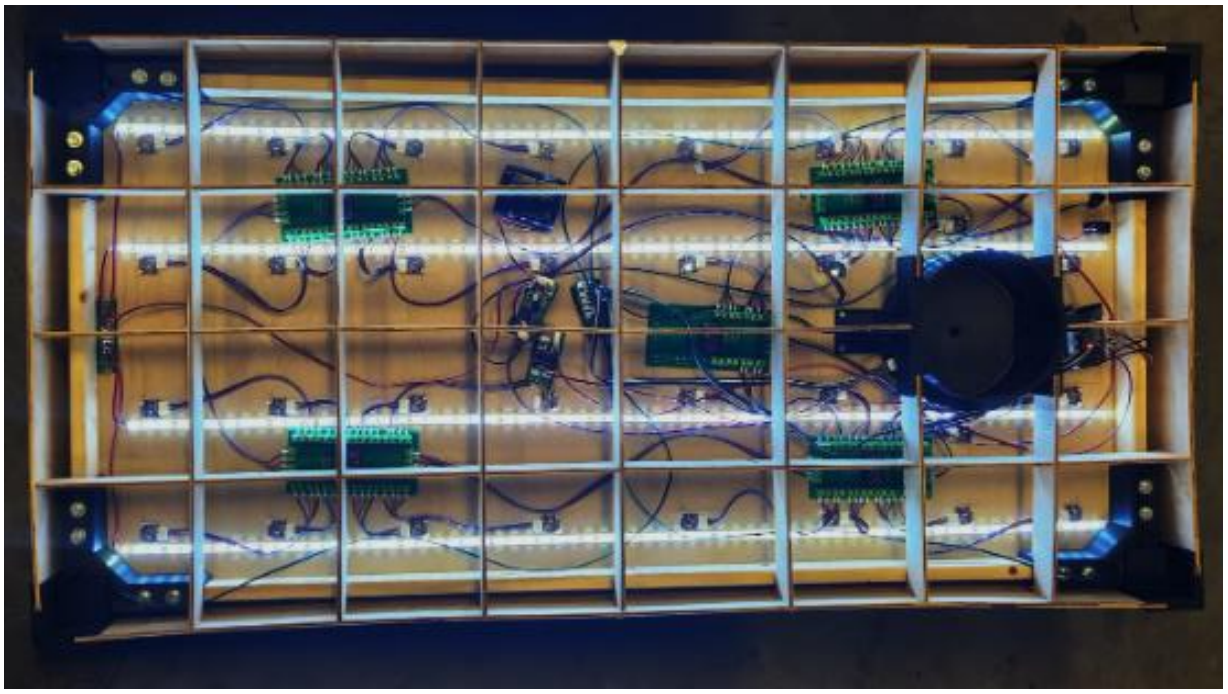


Figure 10.1: Full-Sized Board with LEDs

The group was also able to control the solenoid with a high or low value being sent to it from a GPIO pin. Whenever the solenoid specific GPIO pin was high it would activate the solenoid to retract which released the trap door of the hole and would let the bag drop. Wireless communication was also a big accomplishment of this project. One ESP32 was established as the “primary”, and another was established as the “secondary”. The primary ESP32 was able to send changing data to the secondary ESP32. The data was then displayed using the serial monitor in the Arduino IDE.

A huge part of this project is to be able to control the game and keep the score correctly and the group was able to verify their gameplay logic using artificial numbers. These artificial values were used because values could not be found from the sensors on the full-sized board. This will be explained in more detail in section 10.2. Score values were plugged into the code to increase the score of both teams, but team 1 would increase faster. The code was able to track both team's score through a singular round, cancel out the scores, and hold those values to the remainder of the game until there was a winner.

10.2. Problems Faced

Even though there were many accomplishments from this project, there were three main problems that the group ran into. The first is that the color sensors did not respond when they were trying to be located. This didn't allow the group to test the color sensors on the full-sized board. The second problem was similar to the first, but it involves the pressure sensors. They either gave values of 0 or the full 12-bit value of 4095, this 12-bit value was the analog resolution in the code. This caused the issue of not being able to identify a bag has hit the board let alone located it. The last problem that was identified was that the solenoid was not powerful enough to open the trap door whenever there was a bag resting on it.

11. What Could be Changed?

There are some recommendations that the group believes could fix the problems that were experienced. The reason that the color sensors had issues responding was because there was an immense amount of capacitance along the I2C bus. Since this value of the output was too high, the threshold for the secondary devices won't be reached. This means that they will not recognize the signal at all. [11] The next step would be to try and find a way to reduce this capacitance to be able to have all the color sensors respond.

Another recommendation is to use an op-amp on the pressure sensors to amplify the response. This should allow the values of the pressure sensors to be read. A stronger solenoid should fix the problem of not being able to retract and open the trap door.

The group would also recommend trying another approach if possible. The one approach that would seem to solve issue of this project as well as the previous ones would be to use RFID chips with the optimized pressure sensors. This would allow the bags to be identified and it will also not misidentify bags that missed the board and count them as if they are on the board by having the pressure sensors sense a change in equilibrium.

12. Project Planning

A project such as this, especially starting from scratch, needs to be well planned out. Sections 12.1 and 12.2 will show the bill of materials and the timeline of the project respectfully.

12.1. Bill of Materials

The bill of materials (Table 12.1) is a table that lists out the parts and components that will go into the physical design of the cornhole board. The main components of our system are listed in the system's subsections. The board plans to have 32 color sensors which is roughly 11 units of a 3-pack bundle of color sensors. Four pressure sensors are used, one in each corner. One e-ink display as the physical scoreboard. Two battery packs will be used to power all the systems. One microcontroller will be used to have all the components communicate with each other. Extra LED strips will be used to provide more light intensity for the color sensors to give a consistent reading of whichever color they see.

Table 12.1: Bill of Materials

Item	Qty	Unit Cost	Cost	Description
1	12	$12.88	$154.56	3 pack Color Sensors
2	5	$7.90	$39.50	Pressure Sensor
3	2	$34.99	$69.98	E-ink Display

4	1	$18.95	$18.95	Battery Pack
5	1	$17.99	$17.99	Microcontroller
6	5	$7.92	$39.60	16' LED Strips
7	1	$36.23	$36.23	Plexiglass Cover
8	1	$12.79	$12.79	10 pack Multiplexers
9	3	$1.49	$4.47	Op-Amp
10	1	$3.93	$3.93	Differential Amplifier
11	1	$13.00	$13.00	1/4" Plywood
12	2	$3.00	$6.00	1x2
13	1	$3.00	$3.00	2x4
14	1	$8.49	$8.49	Solenoid
15	5	$1.75	$8.75	Printed Circuit Board
16	100	$0.70	$70.00	Screw-in Terminals
17	1	$18.00	$18.00	3D Filament
18	1	$8.99	$8.99	Buck Boost Converter
19	1	$6.49	$6.49	JST Connectors
20	2	$16.99	$33.98	Ribbon Cable
21	1	$24.99	$24.99	Ferrule Connectors
		Total	**$574.70**	

12.2. Timeline

The specific tasks can be seen in Appendix A. The table shows when those specific tasks were completed. Even in this short amount of time, the group was able to get a good amount of progress done on their project.

13. Conclusion

This idea of a project is to allow players to play the game of cornhole leisurely and not have to worry about the score. It would also allow professional/competitive players to withdraw their data and analyze their performance. This design offers accurate, automated scoring and data with a board and bags that are regulation size and weight. Although an expansion from the prototype did not yield the results that were expected, it was still a valuable start to a project. Not a lot of time was left for implementation and testing the full-sized board due to constructing the physical board as well as distributing the power. This project was still a great, on-hand experience with a lot of learning. This report has the honest results of the testing and prototypes which follows the Code of Ethics for Engineers provided by the National Society of Professional Engineers (NSPE).

References

[1] 07 May 2021. [Online]. Available: https://cornholecanvas.com/blogs/cornhole-life/history-of-cornhole-test-2#:~:text=In%201883%2C%20Heyliger%20de%20Windt,with%20a%20hole%20in%20it..

[2] ACL, "ACL," [Online]. Available: https://www.google.com/imgres?imgurl=https%3A%2F%2Fapp.iplayacl.com%2Fassets%2Ficons%2Ficon-512x512.png&imgrefurl=https%3A%2F%2Fapp.iplayacl.com%2F&tbnid=eRir5mk8RsH8AM&vet=12ahUKEwjzuMXP2bv9AhUG58kDHYgBDTUQMygDegUIARCSAg..i&docid=9RYhcV7TD-aEUM&w=512&h=5. [Accessed 3 2023].

[3] "Amazon," [Online]. Available: http://www.amazon.com.

[4] F. D. A. L. N. P. Diovanni Lara, "Smart Cornhole," 2017. [Online]. Available: https://www.ece.ucf.edu/seniordesign/sp2017su2017/g10/files/Group%2010%20SD1%20Final%20120%20Pages%20Smart%20Cornhole.pdf. [Accessed 2023].

[5] M. B. D. H. Harrison Overturf, "Automatic Score Tracking Cornhole Game," 2020. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1521&context=eesp. [Accessed 2023].

[6] Fantastic Offense, "Dimensions.com," [Online]. Available: https://www.dimensions.com/element/cornhole-bean-bag-toss-boards-platforms. [Accessed April 2023].

[7] "Digikey," [Online]. Available: http://www.digikey.com.

[8] "Upesy," 18 08 2022. [Online]. Available: https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide.

[9] Tosso.com, "How to play cornhole," [Online]. Available: https://www.tosso.com/blogs/news/how-to-play-cornhole. [Accessed March 2023].

[10] Espressif Systems, "Wireless MCUs and AIoT Solutions," [Online]. Available: https://www.espressif.com. [Accessed 9 2023].

[11] K. E. Clothier, "What is Bus capacitance in I2C? How it limits number of devices can be connected to the bus?," Electrical Engineering Stack Exchange, 2020. [Online]. Available: https://electronics.stackexchange.com/questions/494718/what-is-bus-capacitance-in-i2c-how-it-limits-number-of-devices-can-be-connected#:~:text=As%20the%20capacitance%20on%20the,in%20time%20to%20be%20registered.. [Accessed 1 12 2023].

Appendix A: Project Timeline

Task	Finish Date
Prototype Color Sensor Reading	7/30/2023
Prototype Pressure Sensor Calibration	7/30/2023
Full Board Design	9/15/2023
PCB Fabricated	9/27/2023
Version 2 Physical Model Built	10/1/2023
Communication Code	10/13/2023
Version 3 Physical Model Built	10/15/2023
Faculty Presentation	11/7/2023
Gameplay Code	11/15/2023
Implementation and Wiring Complete	11/22/2023
Component Testing	11/30/2023
Testing and Troubleshooting	11/30/2023
Project Presentation	12/1/2023

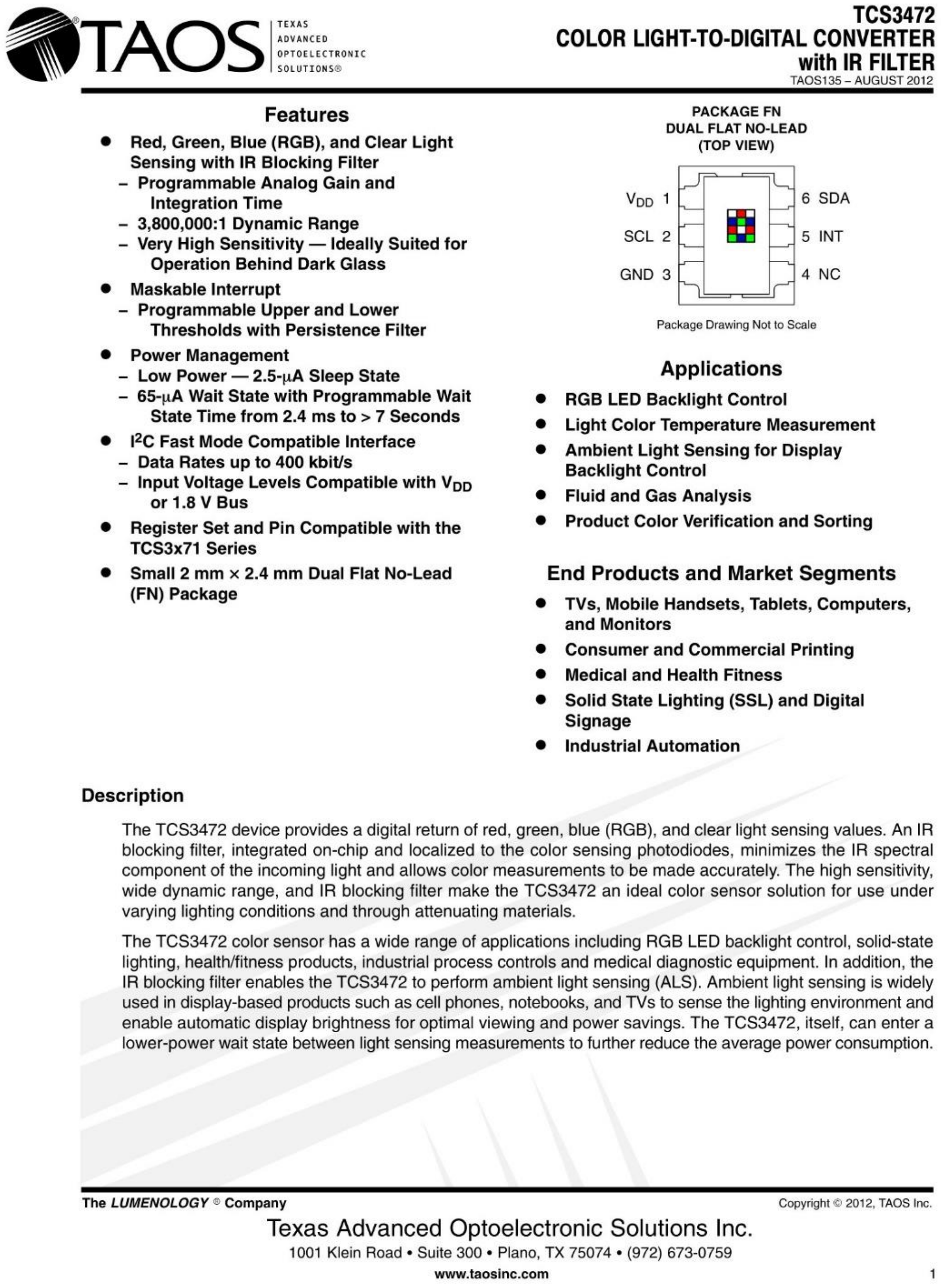
Appendix B: ABET Outcome 2, Design Factor Considerations

ABET Outcome 2 states "*An ability to apply engineering design to produce solutions that meet specified needs with consideration of public health safety, and welfare, as well as global, cultural, social, environmental, and economic factors.*"

ABET also requires that design projects reference appropriate professional standards, such as IEEE, ATSM, etc.

Design Factor	Page number, or reason not applicable
Public health safety, and welfare	Page 20
Global	Page 22
Cultural	Page 10
Social	Page 10
Environmental	Page 20
Economic	Page 22
Ethical & Professional	Page 47
Reference for Standards	NSPE Code of Ethics, Title 47 Federal Communications Commission: Section 15, IEEE 802.11-2012

Appendix C: TCS3472 Datasheet



TAOS TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS®

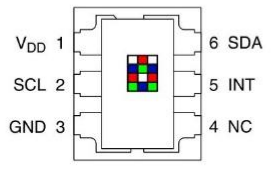
TCS3472
COLOR LIGHT-TO-DIGITAL CONVERTER with IR FILTER

TAOS135 – AUGUST 2012

Features

- Red, Green, Blue (RGB), and Clear Light Sensing with IR Blocking Filter
  - Programmable Analog Gain and Integration Time
  - 3,800,000:1 Dynamic Range
  - Very High Sensitivity — Ideally Suited for Operation Behind Dark Glass
- Maskable Interrupt
  - Programmable Upper and Lower Thresholds with Persistence Filter
- Power Management
  - Low Power — 2.5-μA Sleep State
  - 65-μA Wait State with Programmable Wait State Time from 2.4 ms to > 7 Seconds
- I^{2}C Fast Mode Compatible Interface
  - Data Rates up to 400 kbit/s
  - Input Voltage Levels Compatible with V_{DD} or 1.8 V Bus
- Register Set and Pin Compatible with the TCS3x71 Series
- Small 2 mm × 2.4 mm Dual Flat No-Lead (FN) Package

PACKAGE FN
DUAL FLAT NO-LEAD
(TOP VIEW)



Package Drawing Not to Scale

Applications

- RGB LED Backlight Control
- Light Color Temperature Measurement
- Ambient Light Sensing for Display Backlight Control
- Fluid and Gas Analysis
- Product Color Verification and Sorting

End Products and Market Segments

- TVs, Mobile Handsets, Tablets, Computers, and Monitors
- Consumer and Commercial Printing
- Medical and Health Fitness
- Solid State Lighting (SSL) and Digital Signage
- Industrial Automation

Description

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials.

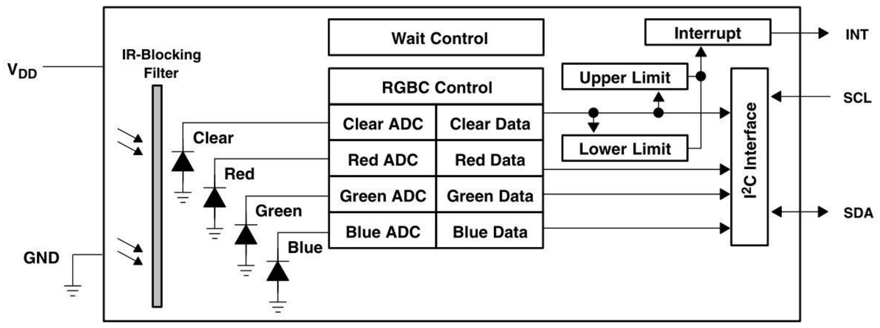
The TCS3472 color sensor has a wide range of applications including RGB LED backlight control, solid-state lighting, health/fitness products, industrial process controls and medical diagnostic equipment. In addition, the IR blocking filter enables the TCS3472 to perform ambient light sensing (ALS). Ambient light sensing is widely used in display-based products such as cell phones, notebooks, and TVs to sense the lighting environment and enable automatic display brightness for optimal viewing and power savings. The TCS3472, itself, can enter a lower-power wait state between light sensing measurements to further reduce the average power consumption.

The *LUMENOLOGY* © Company

Texas Advanced Optoelectronic Solutions Inc.
1001 Klein Road • Suite 300 • Plano, TX 75074 • (972) 673-0759
www.taosinc.com

1

Functional Block Diagram



Detailed Description

The TCS3472 light-to-digital converter contains a 3 × 4 photodiode array, four analog-to-digital converters (ADC) that integrate the photodiode current, data registers, a state machine, and an I^{2}C interface. The 3 × 4 photodiode array is composed of red-filtered, green-filtered, blue-filtered, and clear (unfiltered) photodiodes. In addition, the photodiodes are coated with an IR-blocking filter. The four integrating ADCs simultaneously convert the amplified photodiode currents to a 16-bit digital value. Upon completion of a conversion cycle, the results are transferred to the data registers, which are double-buffered to ensure the integrity of the data. All of the internal timing, as well as the low-power wait state, is controlled by the state machine.

Communication of the TCS3472 data is accomplished over a fast, up to 400 kHz, two-wire I^{2}C serial bus. The industry standard I^{2}C bus facilitates easy, direct connection to microcontrollers and embedded processors.

In addition to the I^{2}C bus, the TCS3472 provides a separate interrupt signal output. When interrupts are enabled, and user-defined thresholds are exceeded, the active-low interrupt is asserted and remains asserted until it is cleared by the controller. This interrupt feature simplifies and improves the efficiency of the system software by eliminating the need to poll the TCS3472. The user can define the upper and lower interrupt thresholds and apply an interrupt persistence filter. The interrupt persistence filter allows the user to define the number of consecutive out-of-threshold events necessary before generating an interrupt. The interrupt output is open-drain, so it can be wire-ORed with other devices.

Terminal Functions

TERMINAL		TYPE	DESCRIPTION
NAME	NO.		
GND	3		Power supply ground. All voltages are referenced to GND.
INT	5	O	Interrupt — open drain (active low).
NC	4	O	No connect — do not connect.
SCL	2	I	I^2C serial clock input terminal — clock signal for I^2C serial data.
SDA	6	I/O	I^2C serial data I/O terminal — serial data I/O for I^2C .
V_{DD}	1		Supply voltage.

Available Options

DEVICE	ADDRESS	PACKAGE – LEADS	INTERFACE DESCRIPTION	ORDERING NUMBER
TCS34721†	0x39	FN–6	I^2C Vbus = V_{DD} Interface	TCS34721FN
TCS34723†	0x39	FN–6	I^2C Vbus = 1.8 V Interface	TCS34723FN
TCS34725	0x29	FN–6	I^2C Vbus = V_{DD} Interface	TCS34725FN
TCS34727	0x29	FN–6	I^2C Vbus = 1.8 V Interface	TCS34727FN

† Contact TAOS for availability.

Absolute Maximum Ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{DD} (Note 1)	3.8 V
Input terminal voltage	–0.5 V to 3.8 V
Output terminal voltage	–0.5 V to 3.8 V
Output terminal current	–1 mA to 20 mA
Storage temperature range, T_{stg}	–40°C to 85°C
ESD tolerance, human body model	2000 V

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltages are with respect to GND.

Recommended Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (TCS34721 & TCS34725) (I^2C V_{bus} = V_{DD})	2.7	3	3.6	V
Supply voltage, V_{DD} (TCS34723 & TCS34727) (I^2C V_{bus} = 1.8 V)	2.7	3	3.3	V
Operating free-air temperature, T_A	–30		70	°C

Operating Characteristics, V_{DD} = 3 V, T_A = 25°C (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD}	Supply current	Active		235	330	μA
		Wait state		65		
		Sleep state — no I^2C activity		2.5	10	
V_{OL}	INT, SDA output low voltage	3 mA sink current	0		0.4	V
		6 mA sink current	0		0.6	
I_{LEAK}	Leakage current, SDA, SCL, INT pins		–5		5	μA
I_{LEAK}	Leakage current, LDR pin		–5		5	μA
V_{IH}	SCL, SDA input high voltage	TCS34721 & TCS34725	0.7 V_{DD}			V
		TCS34723 & TCS34727	1.25			
V_{IL}	SCL, SDA input low voltage	TCS34721 & TCS34725			0.3 V_{DD}	V
		TCS34723 & TCS34727			0.54	

Optical Characteristics, V_{DD} = 3 V, T_A = 25°C, AGAIN = 16×, ATIME = 0xF6 (unless otherwise noted) (Note 1)

PARAMETER		TEST CONDITIONS	Red Channel			Green Channel			Blue Channel			Clear Channel			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
R_e	Irradiance responsivity	λ_D = 465 nm Note 2	0%		15%	10%		42%	65%		88%	11.0	13.8	16.6	counts/ μW/ cm^2
		λ_D = 525 nm Note 3	4%		25%	60%		85%	10%		45%	13.2	16.6	20.0	
		λ_D = 615 nm Note 4	80%		110%	0%		14%	5%		24%	15.6	19.5	23.4	

NOTES: 1. The percentage shown represents the ratio of the respective red, green, or blue channel value to the clear channel value.
2. The 465 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 465 nm, spectral halfwidth $\Delta\lambda½$ = 22 nm.
3. The 525 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 525 nm, spectral halfwidth $\Delta\lambda½$ = 35 nm.
4. The 615 nm input irradiance is supplied by a AlInGaP light-emitting diode with the following characteristics: dominant wavelength λ_D = 615 nm, spectral halfwidth $\Delta\lambda½$ = 15 nm.

RGBC Characteristics, VDD = 3 V, TA = 25°C, AGAIN = 16×, AEN = 1 (unless otherwise noted)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Dark ADC count value	E_e = 0, AGAIN = 60×, ATIME = 0xD6 (100 ms)	0	1	5	counts
ADC integration time step size	ATIME = 0xFF	2.27	2.4	2.56	ms
ADC number of integration steps (Note 5)		1		256	steps
ADC counts per step (Note 5)		0		1024	counts
ADC count value (Note 5)	ATIME = 0xC0 (153.6 ms)	0		65535	counts
Gain scaling, relative to 1× gain setting	4×	3.8	4	4.2	×
	16×	15	16	16.8	
	60×	58	60	63	

NOTE 5: Parameter ensured by design and is not tested.

Wait Characteristics, V_{DD} = 3 V, T_A = 25°C, WEN = 1 (unless otherwise noted)

PARAMETER	TEST CONDITIONS	CHANNEL	MIN	TYP	MAX	UNIT
Wait step size	WTIME = 0xFF		2.27	2.4	2.56	ms
Wait number of integration steps (Note 1)			1		256	steps

NOTE 1: Parameter ensured by design and is not tested.

AC Electrical Characteristics, V_{DD} = 3 V, T_A = 25°C (unless otherwise noted)

PARAMETER†		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$f_{(SCL)}$	Clock frequency (I^2C only)		0		400	kHz
$t_{(BUF)}$	Bus free time between start and stop condition		1.3			μs
$t_{(HDSTA)}$	Hold time after (repeated) start condition. After this period, the first clock is generated.		0.6			μs
$t_{(SUSTA)}$	Repeated start condition setup time		0.6			μs
$t_{(SUSTO)}$	Stop condition setup time		0.6			μs
$t_{(HDDAT)}$	Data hold time		0			μs
$t_{(SUDAT)}$	Data setup time		100			ns
$t_{(LOW)}$	SCL clock low period		1.3			μs
$t_{(HIGH)}$	SCL clock high period		0.6			μs
t_F	Clock/data fall time				300	ns
t_R	Clock/data rise time				300	ns
C_i	Input pin capacitance				10	pF

† Specified by design and characterization; not production tested.

PARAMETER MEASUREMENT INFORMATION

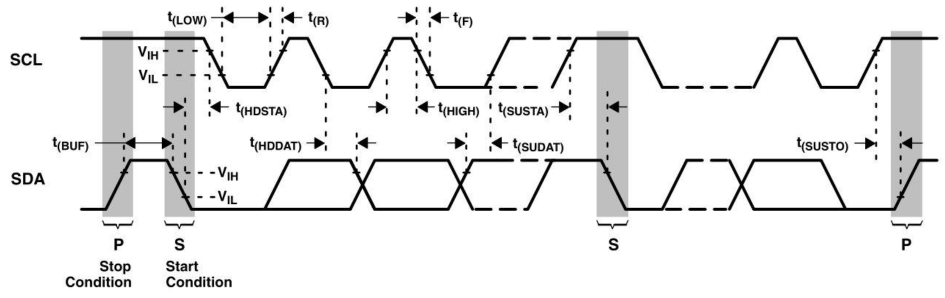


Figure 1. Timing Diagrams

TYPICAL CHARACTERISTICS

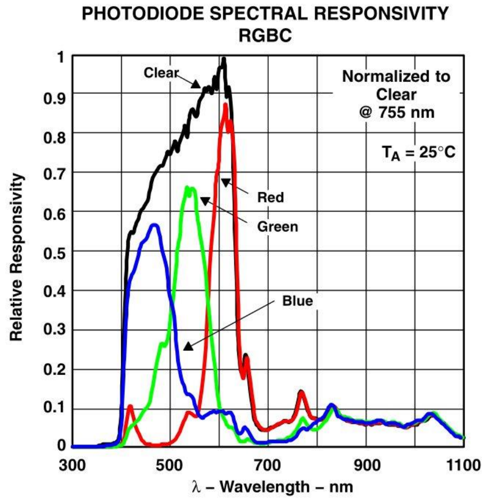


Figure 2

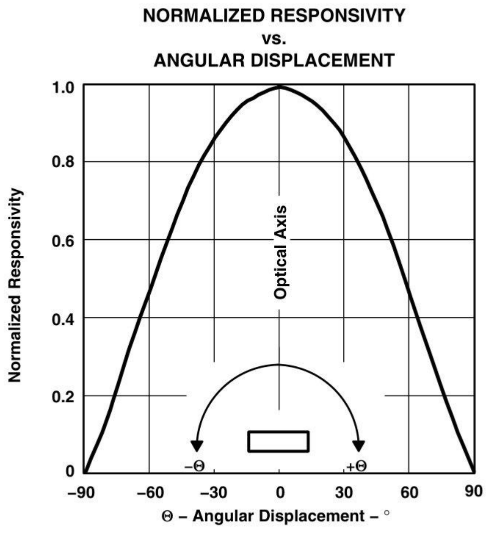


Figure 3

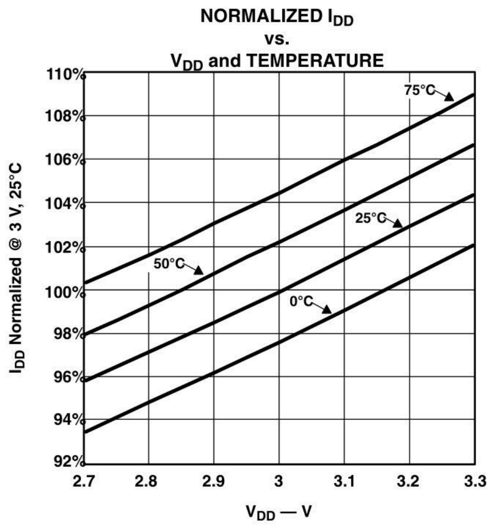


Figure 4

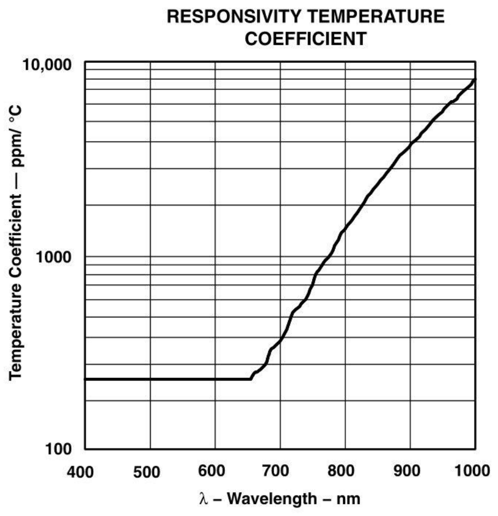


Figure 5

PRINCIPLES OF OPERATION

System States

An internal state machine provides system control of the RGBC and power management features of the device. At power up, an internal power-on-reset initializes the device and puts it in a low-power Sleep state.

When a start condition is detected on the I^2C bus, the device transitions to the Idle state where it checks the Enable Register (0x00) PON bit. If PON is disabled, the device will return to the Sleep state to save power. Otherwise, the device will remain in the Idle state until the RGBC function is enabled (AEN). Once enabled, the device will execute the Wait and RGBC states in sequence as indicated in Figure 5. Upon completion and return to Idle, the device will automatically begin a new Wait-RGBC cycle as long as PON and AEN remain enabled.

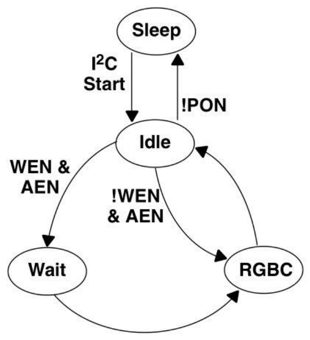


Figure 6. Simplified State Diagram

RGBC Operation

The RGBC engine contains RGBC gain control (AGAIN) and four integrating analog-to-digital converters (ADC) for the RGBC photodiodes. The RGBC integration time (ATIME) impacts both the resolution and the sensitivity of the RGBC reading. Integration of all four channels occurs simultaneously and upon completion of the conversion cycle, the results are transferred to the color data registers. This data is also referred to as channel *count*.

The transfers are double-buffered to ensure that invalid data is not read during the transfer. After the transfer, the device automatically moves to the next state in accordance with the configured state machine.

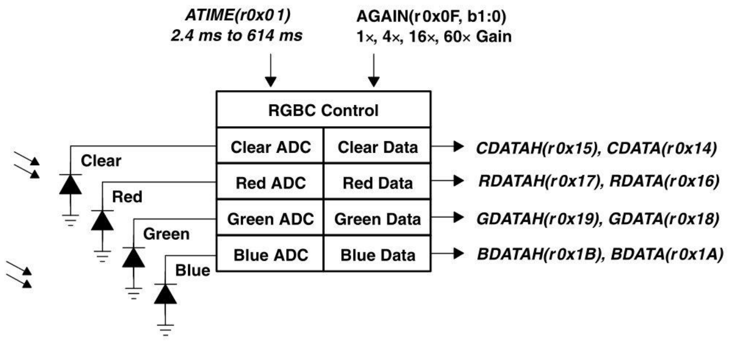


Figure 7. RGBC Operation

NOTE: In this document, the nomenclature uses the bit field name in italics followed by the register address and bit number to allow the user to easily identify the register and bit that controls the function. For example, the power on (PON) is in register 0x00, bit 0. This is represented as *PON (r0x00:b0)*.

The registers for programming the integration and wait times are a 2's compliment values. The actual time can be calculated as follows:

ATIME = 256 − Integration Time / 2.4 ms

Inversely, the time can be calculated from the register value as follows:

Integration Time = 2.4 ms × (256 − ATIME)

For example, if a 100-ms integration time is needed, the device needs to be programmed to:

256 − (100 / 2.4) = 256 − 42 = 214 = 0xD6

Conversely, the programmed value of 0xC0 would correspond to:

(256 − 0xC0) × 2.4 = 64 × 2.4 = 154 ms.

Interrupts

The interrupt feature simplifies and improves system efficiency by eliminating the need to poll the sensor for light intensity values outside of a user-defined range. While the interrupt function is always enabled and its status is available in the status register (0x13), the output of the interrupt state can be enabled using the RGBC interrupt enable (AIEN) field in the enable register (0x00).

Two 16-bit interrupt threshold registers allow the user to set limits below and above a desired light level. An interrupt can be generated when the Clear data (CDATA) is less than the Clear interrupt low threshold (AILTx) or is greater than the Clear interrupt high threshold (AIHTx).

It is important to note that the thresholds are evaluated in sequence, first the low threshold, then the high threshold. As a result, if the low threshold is set above the high threshold, the high threshold is ignored and only the low threshold is evaluated.

To further control when an interrupt occurs, the device provides a persistence filter. The persistence filter allows the user to specify the number of consecutive out-of-range Clear occurrences before an interrupt is generated. The persistence filter register (0x0C) allows the user to set the Clear persistence filter (APERS) value. See the persistence filter register for details on the persistence filter value. Once the persistence filter generates an interrupt, it will continue until a special function interrupt clear command is received (see command register).

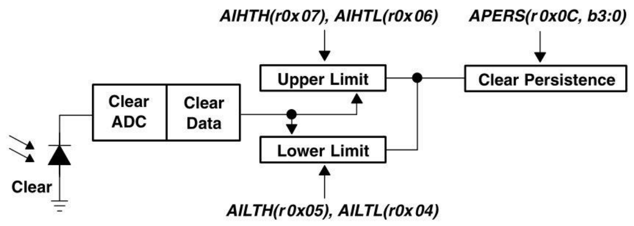


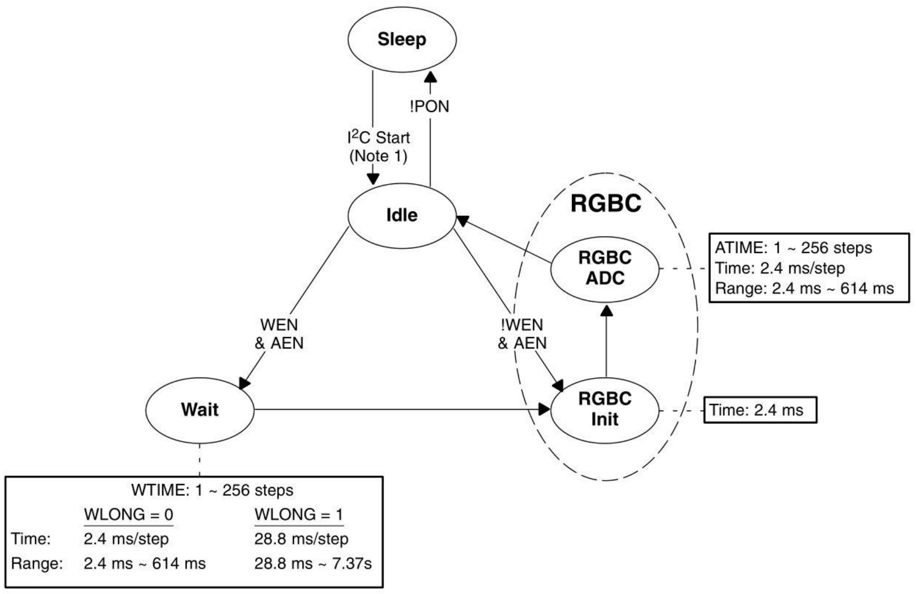
Figure 8. Programmable Interrupt

System Timing

The system state machine shown in Figure 5 provides an overview of the states and state transitions that provide system control of the device. This section highlights the programmable features, which affect the state machine cycle time, and provides details to determine system level timing.

When the power management feature is enabled (WEN), the state machine will transition to the Wait state. The wait time is determined by WLONG, which extends normal operation by 12× when asserted, and WTIME. The formula to determine the wait time is given in the box associated with the Wait state in Figure 9.

When the RGBC feature is enabled (AEN), the state machine will transition through the RGBC Init and RGBC ADC states. The RGBC Init state takes 2.4 ms, while the RGBC ADC time is dependent on the integration time (ATIME). The formula to determine RGBC ADC time is given in the associated box in Figure 9. If an interrupt is generated as a result of the RGBC cycle, it will be asserted at the end of the RGBC ADC.



Notes: 1. There is a 2.4 ms warm-up delay if PON is enabled. If PON is not enabled, the device will return to the Sleep state as shown.
2. PON, WEN, and AEN are fields in the Enable register (0x00).

Figure 9. Detailed State Diagram

Power Management

Power consumption can be managed with the Wait state, because the Wait state typically consumes only 65 μA of I_{DD} current. An example of the power management feature is given below. With the assumptions provided in the example, average I_{DD} is estimated to be 152 μA.

Table 1. Power Management

SYSTEM STATE MACHINE STATE	PROGRAMMABLE PARAMETER	PROGRAMMED VALUE	DURATION	TYPICAL CURRENT
Wait	WTIME	0xEE	43.2 ms	0.065 mA
	WLONG	0		
RGBC Init			2.40 ms	0.235 mA
RGBC ADC	ATIME	0xEE	43.2 ms	0.235 mA

Average I_{DD} Current = $((43.2 \times 0.065) + (43.2 \times 0.235) + (2.40 \times 0.235)) / 89 \approx 152$ μA

Keeping with the same programmed values as the example, Table 2 shows how the average I_{DD} current is affected by the Wait state time, which is determined by WEN, WTIME, and WLONG. Note that the worst-case current occurs when the Wait state is not enabled.

Table 2. Average I_{DD} Current

WEN	WTIME	WLONG	WAIT STATE	AVERAGE I_{DD} CURRENT
0	n/a	n/a	0 ms	291 μA
1	0xFF	0	2.40 ms	280 μA
1	0xEE	0	43.2 ms	152 μA
1	0x00	0	614 ms	82 μA
1	0x00	1	7.37 s	67 μA

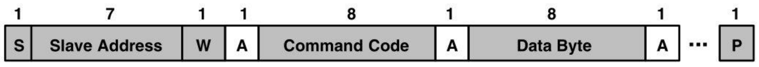
I^{2}C Protocol

Interface and control are accomplished through an I^{2}C serial compatible interface (standard or fast mode) to a set of registers that provide access to device control functions and output data. The devices support the 7-bit I^{2}C addressing protocol.

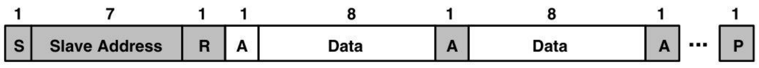
The I^{2}C standard provides for three types of bus transaction: read, write, and a combined protocol (Figure 10). During a write operation, the first byte written is a command byte followed by data. In a combined protocol, the first byte written is the command byte followed by reading a series of bytes. If a read command is issued, the register address from the previous command will be used for data access. Likewise, if the MSB of the command is not set, the device will write a series of bytes at the address stored in the last valid command with a register address. The command byte contains either control information or a 5-bit register address. The control commands can also be used to clear interrupts.

The I^{2}C bus protocol was developed by Philips (now NXP). For a complete description of the I^{2}C protocol, please review the NXP I^{2}C design specification at http://www.i2c−bus.org/references/.

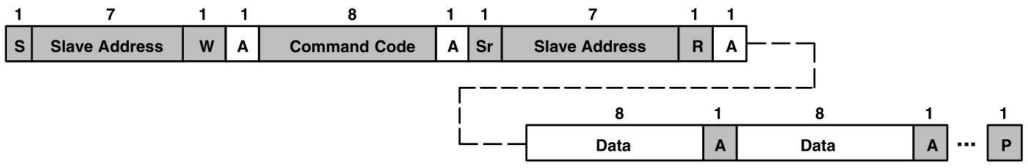
A	Acknowledge (0)
N	Not Acknowledged (1)
P	Stop Condition
R	Read (1)
S	Start Condition
Sr	Repeated Start Condition
W	Write (0)
...	Continuation of protocol
▨	Master-to-Slave
□	Slave-to-Master



I^{2}C Write Protocol



I^{2}C Read Protocol



I^{2}C Read Protocol — Combined Format

Figure 10. I^{2}C Protocols

Register Set

The TCS3472 is controlled and monitored by data registers and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Table 3.

Table 3. Register Address

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
−−	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	RGBC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x05	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filter	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0F	CONTROL	R/W	Control	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATAL	R	Clear data low byte	0x00
0x15	CDATAH	R	Clear data high byte	0x00
0x16	RDATAL	R	Red data low byte	0x00
0x17	RDATAH	R	Red data high byte	0x00
0x18	GDATAL	R	Green data low byte	0x00
0x19	GDATAH	R	Green data high byte	0x00
0x1A	BDATAL	R	Blue data low byte	0x00
0x1B	BDATAH	R	Blue data high byte	0x00

The mechanics of accessing a specific register depends on the specific protocol used. See the section on I^2C protocols on the previous pages. In general, the COMMAND register is written first to specify the specific control-status-data register for subsequent read/write operations.

Command Register

The command register specifies the address of the target register for future write and read operations.

Table 4. Command Register

	7	6	5	4	3	2	1	0	
COMMAND	**CMD**	**TYPE**		**ADDR/SF**					– –

FIELD	BITS	DESCRIPTION	
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.	
TYPE	6:5	Selects type of transaction to follow in subsequent data transfers:	
		FIELD VALUE	**INTEGRATION TIME**
		00	Repeated byte protocol transaction
		01	Auto-increment protocol transaction
		10	Reserved — Do not use
		11	Special function — See description below
		Byte protocol will repeatedly read the same register with each data access. Block protocol will provide auto-increment function to read successive bytes.	
ADDR/SF	4:0	Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below only apply to special function commands:	
		FIELD VALUE	**READ VALUE**
		00110	Clear channel interrupt clear
		other	Reserved — Do not write
		The Clear channel interrupt clear special function clears any pending interrupt and is self-clearing.	

Enable Register (0x00)

The Enable register is used primarily to power the TCS3472 device on and off, and enable functions and interrupts as shown in Table 5.

Table 5. Enable Register

	7	6	5	4	3	2	1	0	
ENABLE	**Reserved**			**AIEN**	**WEN**	**Reserved**	**AEN**	**PON**	**Address 0x00**

FIELD	BITS	DESCRIPTION
Reserved	7:5	Reserved. Write as 0.
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	Reserved. Write as 0.
AEN	1	RGBC enable. This bit actives the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.
PON [1, 2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

NOTES: 1. See Power Management section for more information.
2. A minimum interval of 2.4 ms must pass after PON is asserted before an RGBC can be initiated.

RGBC Timing Register (0x01)

The RGBC timing register controls the internal integration time of the RGBC clear and IR channel ADCs in 2.4-ms increments. Max RGBC Count = (256 − ATIME) × 1024 up to a maximum of 65535.

Table 6. RGBC Timing Register

FIELD	BITS	DESCRIPTION			
ATIME	7:0	**VALUE**	**INTEG_CYCLES**	**TIME**	**MAX COUNT**
		0xFF	1	2.4 ms	1024
		0xF6	10	24 ms	10240
		0xD5	42	101 ms	43008
		0xC0	64	154 ms	65535
		0x00	256	700 ms	65535

Wait Time Register (0x03)

Wait time is set 2.4 ms increments unless the WLONG bit is asserted, in which case the wait times are 12× longer. WTIME is programmed as a 2's complement number.

Table 7. Wait Time Register

FIELD	BITS	DESCRIPTION			
WTIME	7:0	**REGISTER VALUE**	**WAIT TIME**	**TIME (WLONG = 0)**	**TIME (WLONG = 1)**
		0xFF	1	2.4 ms	0.029 sec
		0xAB	85	204 ms	2.45 sec
		0x00	256	614 ms	7.4 sec

RGBC Interrupt Threshold Registers (0x04 – 0x07)

The RGBC interrupt threshold registers provides the values to be used as the high and low trigger points for the comparison function for interrupt generation. If the value generated by the clear channel crosses below the lower threshold specified, or above the higher threshold, an interrupt is asserted on the interrupt pin.

Table 8. RGBC Interrupt Threshold Registers

REGISTER	ADDRESS	BITS	DESCRIPTION
AILTL	0x04	7:0	RGBC clear channel low threshold lower byte
AILTH	0x05	7:0	RGBC clear channel low threshold upper byte
AIHTL	0x06	7:0	RGBC clear channel high threshold lower byte
AIHTH	0x07	7:0	RGBC clear channel high threshold upper byte

Persistence Register (0x0C)

The persistence register controls the filtering interrupt capabilities of the device. Configurable filtering is provided to allow interrupts to be generated after each integration cycle or if the integration has produced a result that is outside of the values specified by the threshold register for some specified amount of time.

Table 9. Persistence Register

	7	6	5	4	3	2	1	0	
PERS	Reserved				APERS				Address 0x0C

FIELD	BITS	DESCRIPTION		
PPERS	7:4	Reserved		
APERS	3:0	Interrupt persistence. Controls rate of interrupt to the host processor.		
		FIELD VALUE	MEANING	INTERRUPT PERSISTENCE FUNCTION
		0000	Every	Every RGBC cycle generates an interrupt
		0001	1	1 clear channel value outside of threshold range
		0010	2	2 clear channel consecutive values out of range
		0011	3	3 clear channel consecutive values out of range
		0100	5	5 clear channel consecutive values out of range
		0101	10	10 clear channel consecutive values out of range
		0110	15	15 clear channel consecutive values out of range
		0111	20	20 clear channel consecutive values out of range
		1000	25	25 clear channel consecutive values out of range
		1001	30	30 clear channel consecutive values out of range
		1010	35	35 clear channel consecutive values out of range
		1011	40	40 clear channel consecutive values out of range
		1100	45	45 clear channel consecutive values out of range
		1101	50	50 clear channel consecutive values out of range
		1110	55	55 clear channel consecutive values out of range
		1111	60	60 clear channel consecutive values out of range

Configuration Register (0x0D)

The configuration register sets the wait long time.

Table 10. Configuration Register

	7	6	5	4	3	2	1	0	
CONFIG	Reserved						WLONG	Reserved	Address 0x0D

FIELD	BITS	DESCRIPTION
Reserved	7:2	Reserved. Write as 0.
WLONG	1	Wait Long. When asserted, the wait cycles are increased by a factor 12× from that programmed in the WTIME register.
Reserved	0	Reserved. Write as 0.

Control Register (0x0F)

The Control register provides eight bits of miscellaneous control to the analog block. These bits typically control functions such as gain settings and/or diode selection.

Table 11. Control Register

	7	6	5	4	3	2	1	0	
CONTROL	Reserved						AGAIN		Address 0x0F

FIELD	BITS	DESCRIPTION	
Reserved	7:2	Reserved. Write bits as 0	
AGAIN	1:0	RGBC Gain Control.	
		FIELD VALUE	RGBC GAIN VALUE
		00	1× gain
		01	4× gain
		10	16× gain
		11	60× gain

ID Register (0x12)

The ID Register provides the value for the part number. The ID register is a read-only register.

Table 12. ID Register

	7	6	5	4	3	2	1	0	
ID	ID								Address 0x12

FIELD	BITS	DESCRIPTION	
ID	7:0	Part number identification	0x44 = TCS34721 and TCS34725
			0x4D = TCS34723 and TCS34727

Status Register (0x13)

The Status Register provides the internal status of the device. This register is read only.

Table 13. Status Register

	7	6	5	4	3	2	1	0	
STATUS	**Reserved**			**AINT**	**Reserved**			**AVALID**	**Address 0x13**

FIELD	BIT	DESCRIPTION
Reserved	7:5	Reserved.
AINT	4	RGBC clear channel Interrupt.
Reserved	3:1	Reserved.
AVALID	0	RGBC Valid. Indicates that the RGBC channels have completed an integration cycle.

RGBC Channel Data Registers (0x14 – 0x1B)

Clear, red, green, and blue data is stored as 16-bit values. To ensure the data is read correctly, a two-byte read I^2C transaction should be used with a read word protocol bit set in the command register. With this operation, when the lower byte register is read, the upper eight bits are stored into a shadow register, which is read by a subsequent read to the upper byte. The upper register will read the correct value even if additional ADC integration cycles end between the reading of the lower and upper registers.

Table 14. ADC Channel Data Registers

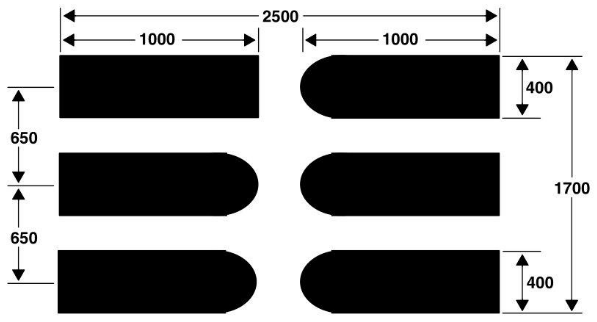
REGISTER	ADDRESS	BITS	DESCRIPTION
CDATA	0x14	7:0	Clear data low byte
CDATAH	0x15	7:0	Clear data high byte
RDATA	0x16	7:0	Red data low byte
RDATAH	0x17	7:0	Red data high byte
GDATA	0x18	7:0	Green data low byte
GDATAH	0x19	7:0	Green data high byte
BDATA	0x1A	7:0	Blue data low byte
BDATAH	0x1B	7:0	Blue data high byte

APPLICATION INFORMATION: HARDWARE

PCB Pad Layout

Suggested PCB pad layout guidelines for the Dual Flat No-Lead (FN) surface mount package are shown in Figure 11.

Note: Pads can be extended further if hand soldering is needed.

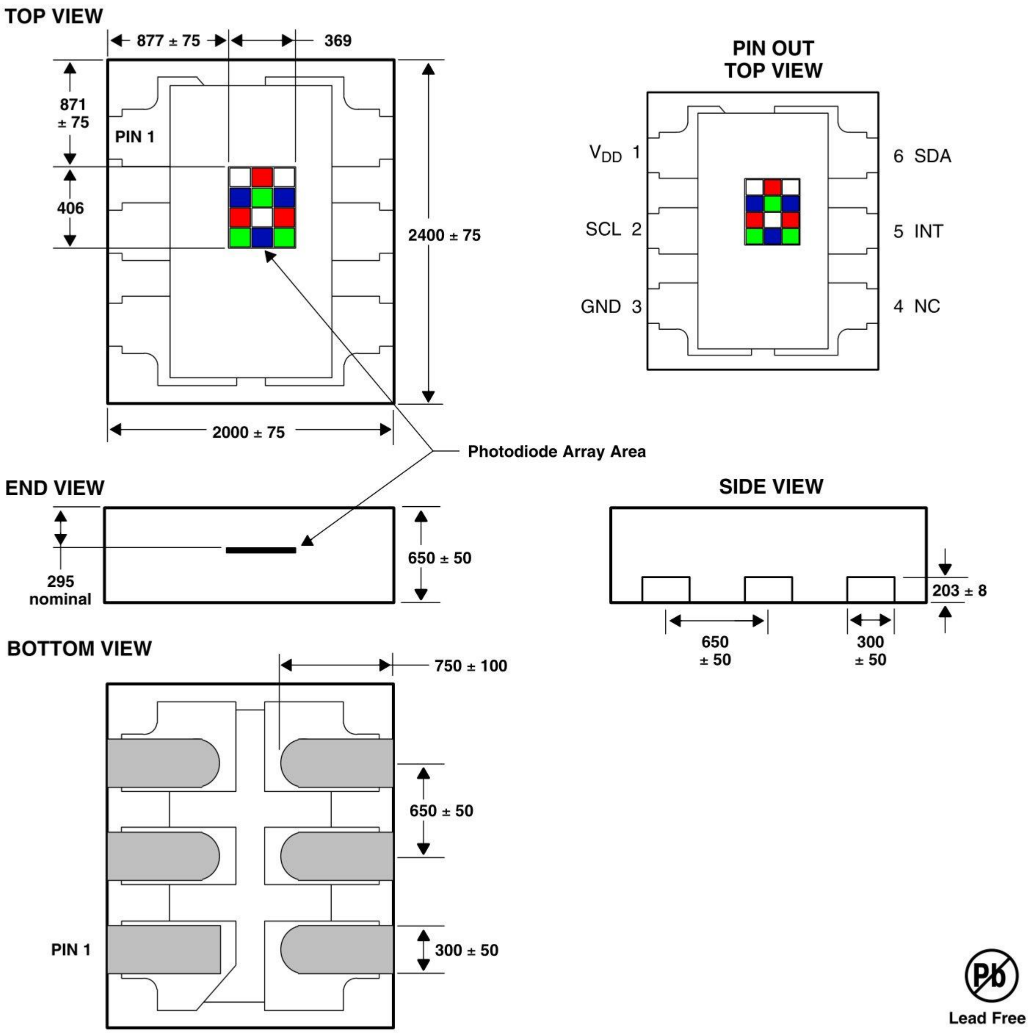


NOTES: A. All linear dimensions are in micrometers.
B. This drawing is subject to change without notice.

Figure 11. Suggested FN Package PCB Layout

PACKAGE INFORMATION

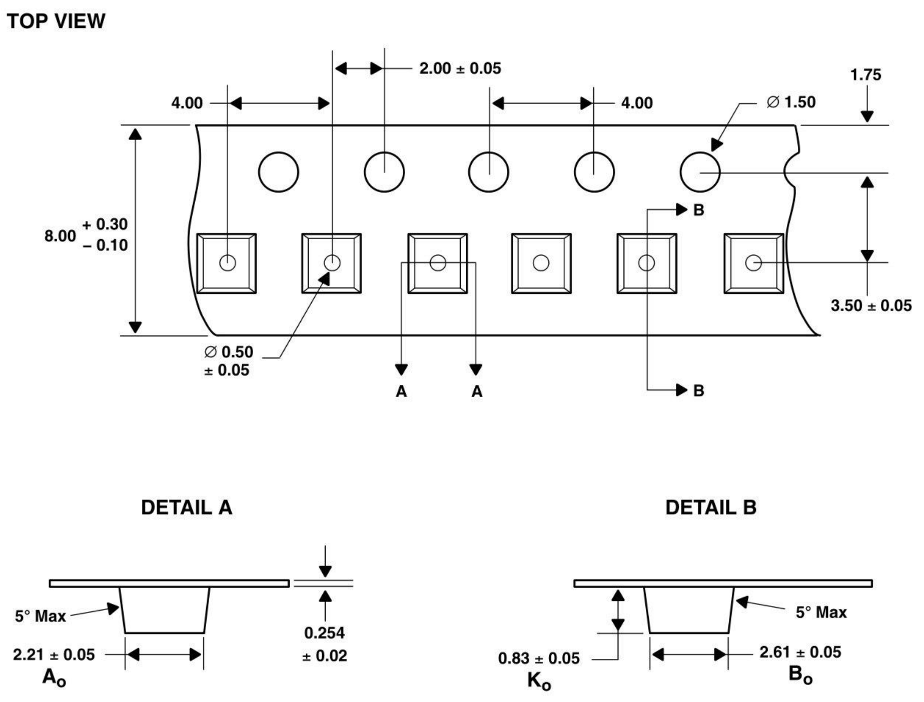
PACKAGE FN **Dual Flat No-Lead**

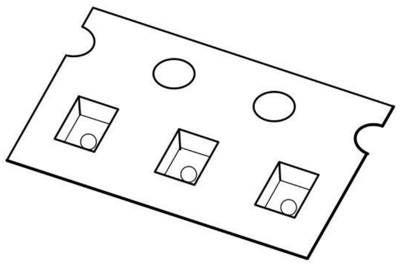


NOTES: A. All linear dimensions are in micrometers. Dimension tolerance is ± 20 μm unless otherwise noted.
B. The die is centered within the package within a tolerance of ± 3 mils.
C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
D. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
E. This package contains no lead (Pb).
F. This drawing is subject to change without notice.

Figure 12. Package FN — Dual Flat No-Lead Packaging Configuration

CARRIER TAPE AND REEL INFORMATION





NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
C. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481−B 2001.
D. Each reel is 178 millimeters in diameter and contains 3500 parts.
E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481−B.
F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
G. This drawing is subject to change without notice.

Figure 13. Package FN Carrier Tape

SOLDERING INFORMATION

The FN package has been tested and has demonstrated an ability to be reflow soldered to a PCB substrate. The process, equipment, and materials used in these test are detailed below.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Table 15. Solder Reflow Profile

PARAMETER	REFERENCE	DEVICE
Average temperature gradient in preheating		2.5°C/sec
Soak time	t_{soak}	2 to 3 minutes
Time above 217°C (T1)	t_1	Max 60 sec
Time above 230°C (T2)	t_2	Max 50 sec
Time above T_{peak} –10°C (T3)	t_3	Max 10 sec
Peak temperature in reflow	T_{peak}	260°C
Temperature gradient in cooling		Max –5°C/sec

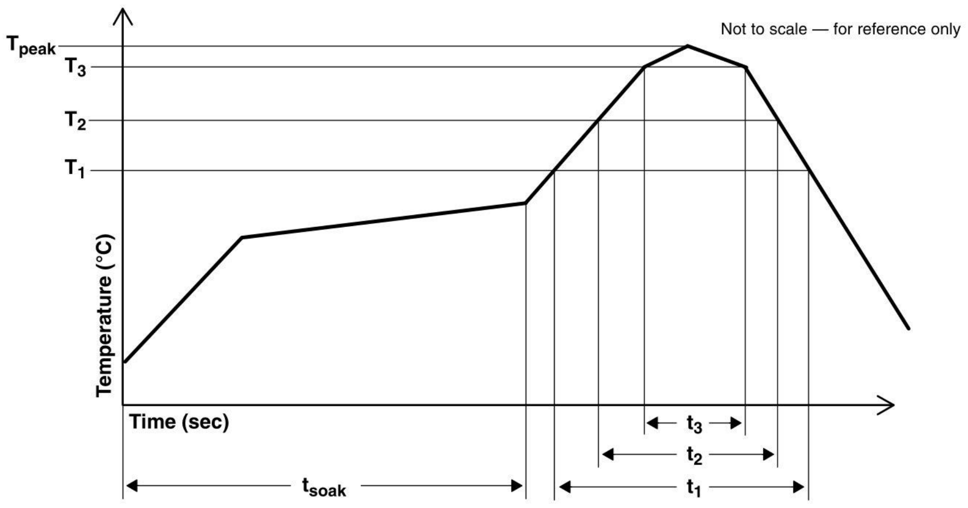


Figure 14. Solder Reflow Profile Graph

STORAGE INFORMATION

Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package. To ensure the package contains the smallest amount of absorbed moisture possible, each device is dry-baked prior to being packed for shipping. Devices are packed in a sealed aluminized envelope called a moisture barrier bag with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

The Moisture Barrier Bags should be stored under the following conditions:

Temperature Range	< 40°C
Relative Humidity	< 90%
Total Time	No longer than 12 months from the date code on the aluminized envelope if unopened.

Rebaking of the reel will be required if the devices have been stored unopened for more than 12 months and the Humidity Indicator Card shows the parts to be out of the allowable moisture region.

Opened reels should be used within 168 hours if exposed to the following conditions:

Temperature Range	< 30°C
Relative Humidity	< 60%

If rebaking is required, it should be done at 50°C for 12 hours.

The FN package has been assigned a moisture sensitivity level of MSL 3.

LEAD-FREE (Pb-FREE) and GREEN STATEMENT

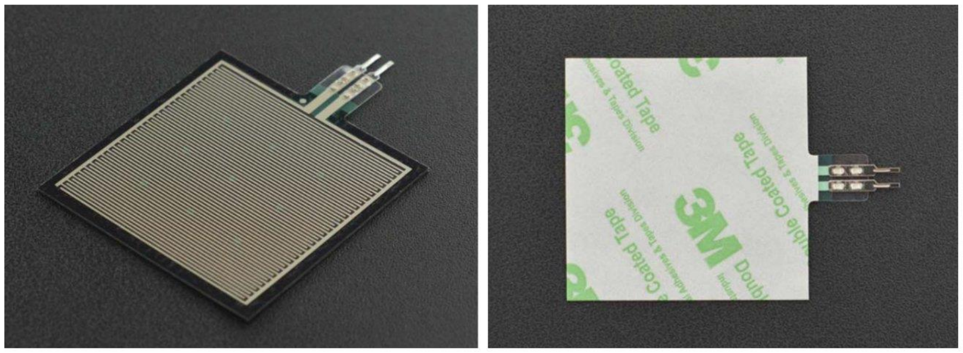
Pb-Free (RoHS) TAOS' terms *Lead-Free* or *Pb-Free* mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TAOS Pb-Free products are suitable for use in specified lead-free processes.

Green (RoHS & no Sb/Br) TAOS defines *Green* to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information and Disclaimer The information provided in this statement represents TAOS' knowledge and belief as of the date that it is provided. TAOS bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TAOS has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TAOS and TAOS suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

NOTICE

Appendix D: SEN0296 Datasheet



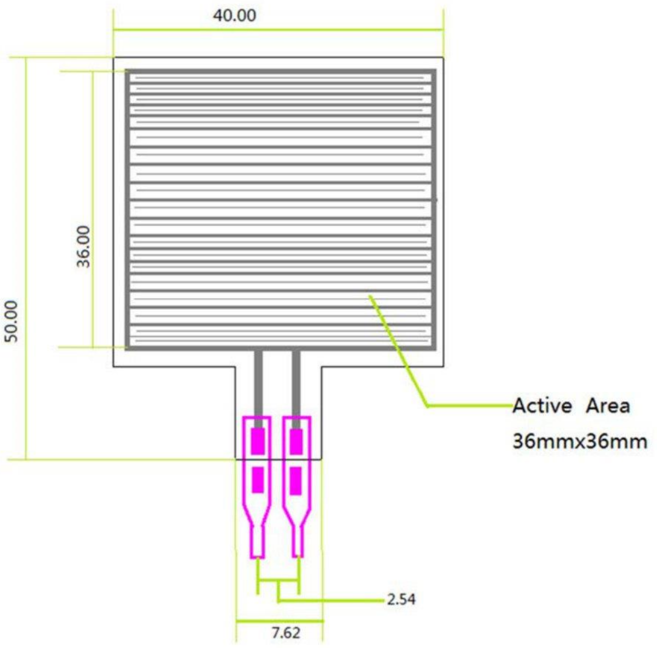
RP-S40-ST Thin Film Pressure Sensor 40mmx40mm

SKU:SEN0296

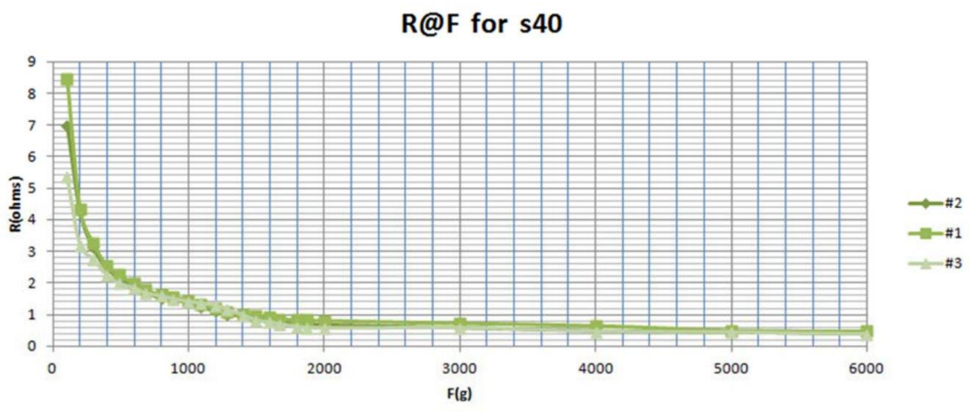
INTRODUCTION

This is a square flexible thin film pressure sensor of short legs with a side length of 40mm, which can be used to realize highly sensitive detection of pressure. The sensor is durable and designed to sense static and dynamic pressure in a high respond speed. Its advantages of recording the intensity and frequency of force make it widely used in all kinds of applications, such as, pressure switch, bed monitoring system, intelligent sneaker and medical device system. These sensors are also very easy to use.

RP-C flexible pressure sensor is made of ultra-thin film of excellent mechanical property, excellent conductive materials and nanometre pressure sensitive layers. There are thin film and pressure sensitive layer on the upper layer of the sensor, and thin film and conductive circuit on the lower layer. These two layers are glued together by double sided tape. When outside pressure applies to the active area, the disconnected circuit of the lower layer will be connected through the pressure sensitive layer of the upper layer, by which to convert pressure into resistance. The output resistance decreases as pressure increases.



RP-S40-ST Dimension Diagram



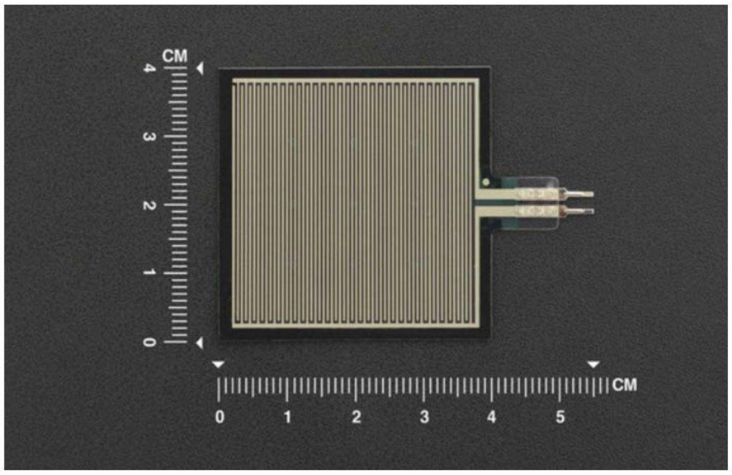
Force vs Resistance

SPECIFICATION

- Thickness: 0.45mm
- Trigger Force: 20g, triggered (default resistance<200kΩ)
- Pressure Measuring Range: 20g~10kg
- Static Pressure & Dynamic Pressure Measurement (within the frequency of 10Hz)
- Initial Resistance: >10MΩ
- Activation Time: <0.01S
- Operating Temperature: -40℃~+85℃
- Lifespan: >1million times
- Hysteresis: +10%, (RF+ -RF-)/FR+, 1000g Force
- Response Time: <10ms
- EMI: Not generate
- EDS: Not generate
- Drift: <5%, 2.5Kg Force , Static load 24H

SHIPPING LIST

- RP-S40-ST Thin Film Pressure Sensor x1



https://www.dfrobot.com/product-1842.html?search=SEN0296/5-9-19

Appendix E: ESP32 Datasheet

ESP32-WROOM-32

Datasheet

Version 3.4
Espressif Systems
Copyright © 2023

www.espressif.com

About This Document

This document provides the specifications for the ESP32-WROOM-32 module.

Document Updates

Please always refer to the latest version on https://www.espressif.com/en/support/download/documents.

Revision History

For revision history of this document, please refer to the last page.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe. Note that you need to update your subscription to receive notifications of new products you are not currently subscribed to.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Contents

1 Overview 6

2 Pin Definitions 8
2.1 Pin Layout 8
2.2 Pin Description 8
2.3 Strapping Pins 10

3 Functional Description 12
3.1 CPU and Internal Memory 12
3.2 External Flash and SRAM 12
3.3 Crystal Oscillators 12
3.4 RTC and Low-Power Management 13

4 Peripherals and Sensors 14

5 Electrical Characteristics 15
5.1 Absolute Maximum Ratings 15
5.2 Recommended Operating Conditions 15
5.3 DC Characteristics (3.3 V, 25 °C) 15
5.4 Wi-Fi Radio 16
5.5 Bluetooth LE Radio 17
5.5.1 Receiver 17
5.5.2 Transmitter 17

6 Schematics 18

7 Peripheral Schematics 19

8 Physical Dimensions 20

9 Recommended PCB Land Pattern 21

10 Product Handling 22
10.1 Storage Conditions 22
10.2 Electrostatic Discharge (ESD) 22
10.3 Reflow Profile 22
10.4 Ultrasonic Vibration 23

11 Related Documentation and Resources 24

Revision History 25

List of Tables

1	ESP32-WROOM-32 Specifications	6
2	Pin Definitions	8
3	Strapping Pins	10
4	Parameter Descriptions of Setup and Hold Times for the Strapping Pins	11
5	Absolute Maximum Ratings	15
6	Recommended Operating Conditions	15
7	DC Characteristics (3.3 V, 25 °C)	15
8	Wi-Fi Radio Characteristics	16
9	Receiver Characteristics – Bluetooth LE	17
10	Transmitter Characteristics – Bluetooth LE	17

List of Figures

1	ESP32-WROOM-32 Pin Layout (Top View)	8
2	Setup and Hold Times for the Strapping Pins	11
3	ESP32-WROOM-32 Schematics	18
4	ESP32-WROOM-32 Peripheral Schematics	19
5	Physical Dimensions of ESP32-WROOM-32	20
6	Recommended PCB Land Pattern	21
7	Reflow Profile	22

1 Overview

ESP32-WROOM-32 is a powerful, generic Wi-Fi + Bluetooth® + Bluetooth LE MCU module that targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding.

At the core of this module is the ESP32-D0WDQ6 chip*. The chip embedded is designed to be scalable and adaptive. There are two CPU cores that can be individually controlled, and the CPU clock frequency is adjustable from 80 MHz to 240 MHz. The chip also has a low-power coprocessor that can be used instead of the CPU to save power while performing tasks that do not require much computing power, such as monitoring of peripherals. ESP32 integrates a rich set of peripherals, ranging from capacitive touch sensors, SD card interface, Ethernet, high-speed SPI, UART, I2S, and I2C.

Note:
* For details on the part numbers of the ESP32 family of chips, please refer to the document *ESP32 Datasheet*.

The integration of Bluetooth, Bluetooth LE and Wi-Fi ensures that a wide range of applications can be targeted, and that the module is all-around: using Wi-Fi allows a large physical range and direct connection to the Internet through a Wi-Fi router, while using Bluetooth allows the user to conveniently connect to the phone or broadcast low energy beacons for its detection. The sleep current of the ESP32 chip is less than 5 μA, making it suitable for battery powered and wearable electronics applications. The module supports a data rate of up to 150 Mbps, and 20 dBm output power at the antenna to ensure the widest physical range. As such the module does offer industry-leading specifications and the best performance for electronic integration, range, power consumption, and connectivity.

The operating system chosen for ESP32 is freeRTOS with LwIP; TLS 1.2 with hardware acceleration is built in as well. Secure (encrypted) over the air (OTA) upgrade is also supported, so that users can upgrade their products even after their release, at minimum cost and effort.

Table 1 provides the specifications of ESP32-WROOM-32.

Table 1: ESP32-WROOM-32 Specifications

Categories	Items	Specifications
Certification	RF certification	See certificates for ESP32-WROOM-32
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Center frequency range of operating channel	2412 ~ 2484 MHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and Bluetooth LE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH

Categories	Items	Specifications
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®), compatible with ISO11898-1 (CAN Specification 2.0)
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating ambient temperature range	–40 °C ~ +85 °C
	Package size	18 mm × 25.5 mm × 3.10 mm
	Moisture sensitivity level (MSL)	Level 3

2 Pin Definitions

2.1 Pin Layout

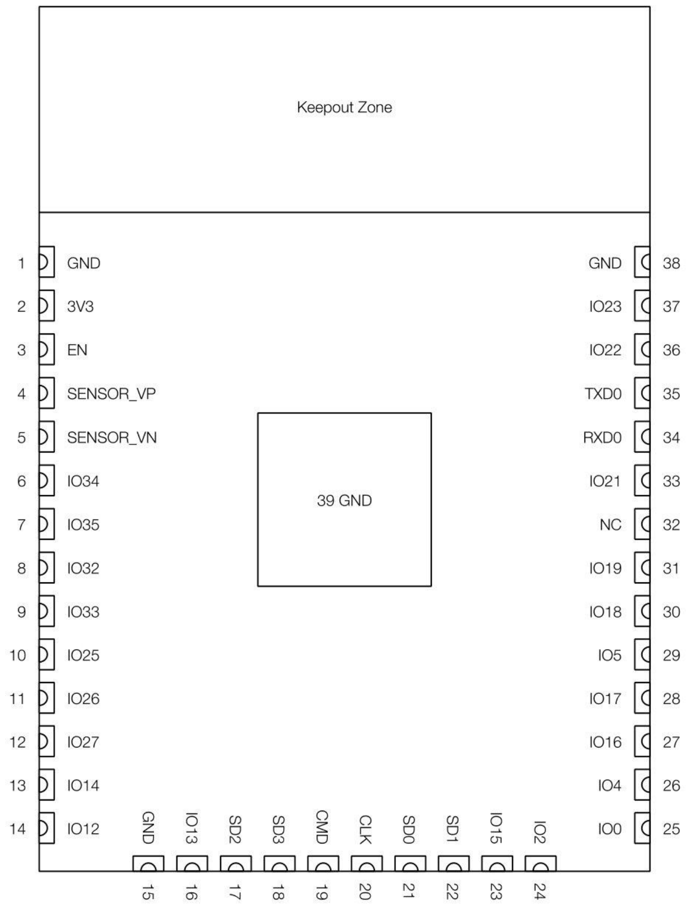


Figure 1: ESP32-WROOM-32 Pin Layout (Top View)

2.2 Pin Description

ESP32-WROOM-32 has 38 pins. See pin definitions in Table 2.

Table 2: Pin Definitions

Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.

Name	No.	Type	Function
SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2*	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3*	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD*	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK*	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0*	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1*	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
IO16	27	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
IO17	28	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

Notice:

* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on the module and are not recommended for other uses.

2.3 Strapping Pins

ESP32 has five strapping pins, which can be seen in Chapter 6 Schematics:

- MTDI
- GPIO0
- GPIO2
- MTDO
- GPIO5

Software can read the values of these five bits from register "GPIO_STRAPPING".

During the chip's system reset release (power-on-reset, RTC watchdog reset and brownout reset), the latches of the strapping pins sample the voltage level as strapping bits of "0" or "1", and hold these bits until the chip is powered down or shut down. The strapping bits configure the device's boot mode, the operating voltage of VDD_SDIO and other initial system settings.

Each strapping pin is connected to its internal pull-up/pull-down during the chip reset. Consequently, if a strapping pin is unconnected or the connected external circuit is high-impedance, the internal weak pull-up/pull-down will determine the default input level of the strapping pins.

To change the strapping bit values, users can apply the external pull-down/pull-up resistances, or use the host MCU's GPIOs to control the voltage level of these pins when powering on ESP32.

After reset release, the strapping pins work as normal-function pins.

Refer to Table 3 for a detailed boot-mode configuration by strapping pins.

Table 3: Strapping Pins

Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3 V		1.8 V	
MTDI	Pull-down	0		1	
Booting Mode					
Pin	Default	SPI Boot		Download Boot	
GPIO0	Pull-up	1		0	
GPIO2	Pull-down	Don't-care		0	
Enabling/Disabling Debugging Log Print over U0TXD During Booting					
Pin	Default	U0TXD Active		U0TXD Silent	
MTDO	Pull-up	1		0	
Timing of SDIO Slave					
Pin	Default	FE Sampling FE Output	FE Sampling RE Output	RE Sampling FE Output	RE Sampling RE Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Note:
- Firmware can configure register bits to change the settings of ”Voltage of Internal LDO (VDD_SDIO)” and ”Timing of SDIO Slave” after booting.
- The module integrates a 3.3 V SPI flash, so the pin MTDI cannot be set to 1 when the module is powered up.

The illustration below shows the setup and hold times for the strapping pins before and after the CHIP_PU signal goes high. Details about the parameters are listed in Table 4.

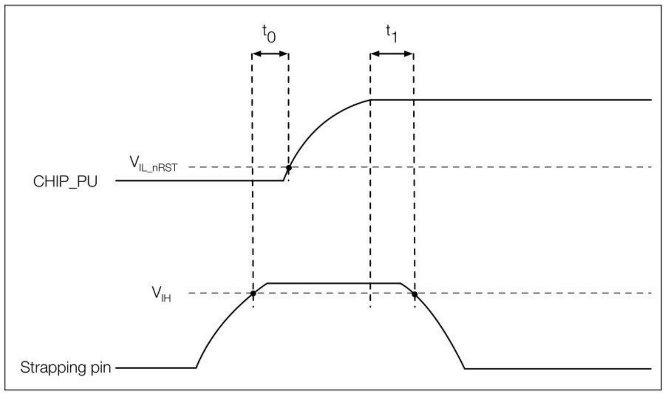


Figure 2: Setup and Hold Times for the Strapping Pins

Table 4: Parameter Descriptions of Setup and Hold Times for the Strapping Pins

Parameters	Description	Min.	Unit
t_0	Setup time before CHIP_PU goes from low to high	0	ms
t_1	Hold time after CHIP_PU goes high	1	ms

3 Functional Description

This chapter describes the modules and functions integrated in ESP32-WROOM-32.

3.1 CPU and Internal Memory

ESP32-D0WDQ6 contains two low-power Xtensa® 32-bit LX6 microprocessors. The internal memory includes:

- 448 KB of ROM for booting and core functions.
- 520 KB of on-chip SRAM for data and instructions.
- 8 KB of SRAM in RTC, which is called RTC FAST Memory and can be used for data storage; it is accessed by the main CPU during RTC Boot from the Deep-sleep mode.
- 8 KB of SRAM in RTC, which is called RTC SLOW Memory and can be accessed by the co-processor during the Deep-sleep mode.
- 1 Kbit of eFuse: 256 bits are used for the system (MAC address and chip configuration) and the remaining 768 bits are reserved for customer applications, including flash-encryption and chip-ID.

3.2 External Flash and SRAM

ESP32 supports multiple external QSPI flash and SRAM chips. More details can be found in Chapter SPI in the *ESP32 Technical Reference Manual*. ESP32 also supports hardware encryption/decryption based on AES to protect developers' programs and data in flash.

ESP32 can access the external QSPI flash and SRAM through high-speed caches.

- The external flash can be mapped into CPU instruction memory space and read-only memory space simultaneously.
  - When external flash is mapped into CPU instruction memory space, up to 11 MB + 248 KB can be mapped at a time. Note that if more than 3 MB + 248 KB are mapped, cache performance will be reduced due to speculative reads by the CPU.
  - When external flash is mapped into read-only data memory space, up to 4 MB can be mapped at a time. 8-bit, 16-bit and 32-bit reads are supported.
- External SRAM can be mapped into CPU data memory space. Up to 4 MB can be mapped at a time. 8-bit, 16-bit and 32-bit reads and writes are supported.

ESP32-WROOM-32 integrates a 4 MB SPI flash, which is connected to GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 and GPIO11. These six pins cannot be used as regular GPIOs.

3.3 Crystal Oscillators

The module uses a 40-MHz crystal oscillator.

3.4 RTC and Low-Power Management

With the use of advanced power-management technologies, ESP32 can switch between different power modes.

For details on ESP32's power consumption in different power modes, please refer to section "RTC and Low-Power Management" in *ESP32 Datasheet*.

4 Peripherals and Sensors

Please refer to Section Peripherals and Sensors in *ESP32 Datasheet*.

Note:
External connections can be made to any GPIO except for GPIOs in the range 6-11. These six GPIOs are connected to the module's integrated SPI flash. For details, please see Section 6 Schematics.

5 Electrical Characteristics

5.1 Absolute Maximum Ratings

Stresses beyond the absolute maximum ratings listed in Table 5 below may cause permanent damage to the device. These are stress ratings only, and do not refer to the functional operation of the device that should follow the recommended operating conditions.

Table 5: Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Unit
VDD33	Power supply voltage	–0.3	3.6	V
I_{output}[1]	Cumulative IO output current	-	1,100	mA
T_{store}	Storage temperature	–40	105	°C

1. The module worked properly after a 24-hour test in ambient temperature at 25 °C, and the IOs in three domains (VDD3P3_RTC, VDD3P3_CPU, VDD_SDIO) output high logic level to ground. Please note that pins occupied by flash and/or PSRAM in the VDD_SDIO power domain were excluded from the test.
2. Please see Appendix IO_MUX of *ESP32 Datasheet* for IO's power domain.

5.2 Recommended Operating Conditions

Table 6: Recommended Operating Conditions

Symbol	Parameter	Min	Typical	Max	Unit
VDD33	Power supply voltage	3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T	Operating ambient temperature	–40	-	85	°C

5.3 DC Characteristics (3.3 V, 25 °C)

Table 7: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter		Min	Typ	Max	Unit
C_{IN}	Pin capacitance		-	2	-	pF
V_{IH}	High-level input voltage		0.75×VDD[1]	-	VDD[1]+0.3	V
V_{IL}	Low-level input voltage		–0.3	-	0.25×VDD[1]	V
I_{IH}	High-level input current		-	-	50	nA
I_{IL}	Low-level input current		-	-	50	nA
V_{OH}	High-level output voltage		0.8×VDD[1]	-	-	V
V_{OL}	Low-level output voltage		-	-	0.1×VDD[1]	V
I_{OH}	High-level source current (VDD[1] = 3.3 V, V_{OH} >= 2.64 V, output drive strength set to the maximum)	VDD3P3_CPU power domain [1, 2]	-	40	-	mA
		VDD3P3_RTC power domain [1, 2]	-	40	-	mA
		VDD_SDIO power domain [1, 3]	-	20	-	mA

Symbol	Parameter	Min	Typ	Max	Unit
I_{OL}	Low-level sink current (VDD[1] = 3.3 V, V_{OL} = 0.495 V, output drive strength set to the maximum)	-	28	-	mA
R_{PU}	Resistance of internal pull-up resistor	-	45	-	kΩ
R_{PD}	Resistance of internal pull-down resistor	-	45	-	kΩ
V_{IL_nRST}	Low-level input voltage of CHIP_PU to shut down the chip	-	-	0.6	V

Notes:

1. Please see Appendix IO_MUX of *ESP32 Datasheet* for IO's power domain. VDD is the I/O voltage for a particular power domain of pins.
2. For VDD3P3_CPU and VDD3P3_RTC power domain, per-pin current sourced in the same domain is gradually reduced from around 40 mA to around 29 mA, V_{OH}>=2.64 V, as the number of current-source pins increases.
3. Pins occupied by flash and/or PSRAM in the VDD_SDIO power domain were excluded from the test.

5.4 Wi-Fi Radio

Table 8: Wi-Fi Radio Characteristics

Parameter	Condition	Min	Typical	Max	Unit
Center frequency range of operating channel ^{note1}	-	2412	-	2484	MHz
Output impedance ^{note2}	-	-	*note 2*	-	Ω
TX power ^{note3}	11n, MCS7	12	13	14	dBm
	11b mode	17.5	18.5	20	dBm
Sensitivity	11b, 1 Mbps	-	–98	-	dBm
	11b, 11 Mbps	-	–89	-	dBm
	11g, 6 Mbps	-	–92	-	dBm
	11g, 54 Mbps	-	–74	-	dBm
	11n, HT20, MCS0	-	–91	-	dBm
	11n, HT20, MCS7	-	–71	-	dBm
	11n, HT40, MCS0	-	–89	-	dBm
	11n, HT40, MCS7	-	–69	-	dBm
Adjacent channel rejection	11g, 6 Mbps	-	31	-	dB
	11g, 54 Mbps	-	14	-	dB
	11n, HT20, MCS0	-	31	-	dB
	11n, HT20, MCS7	-	13	-	dB

1. Device should operate in the center frequency range of operating channel allocated by regional regulatory authorities. Target center frequency range of operating channel is configurable by software.
2. For the modules that use external antennas, the output impedance is 50 Ω. For other modules without external antennas, users do not need to concern about the output impedance.
3. Target TX power is configurable based on device or certification requirements.

5.5 Bluetooth LE Radio

5.5.1 Receiver

Table 9: Receiver Characteristics – Bluetooth LE

Parameter	Conditions	Min	Typ	Max	Unit
Sensitivity @30.8% PER	-	-	–97	-	dBm
Maximum received signal @30.8% PER	-	0	-	-	dBm
Co-channel C/I	-	-	+10	-	dB
Adjacent channel selectivity C/I	F = F0 + 1 MHz	-	–5	-	dB
	F = F0 – 1 MHz	-	–5	-	dB
	F = F0 + 2 MHz	-	–25	-	dB
	F = F0 – 2 MHz	-	–35	-	dB
	F = F0 + 3 MHz	-	–25	-	dB
	F = F0 – 3 MHz	-	–45	-	dB
Out-of-band blocking performance	30 MHz ~ 2000 MHz	–10	-	-	dBm
	2000 MHz ~ 2400 MHz	–27	-	-	dBm
	2500 MHz ~ 3000 MHz	–27	-	-	dBm
	3000 MHz ~ 12.5 GHz	–10	-	-	dBm
Intermodulation	-	–36	-	-	dBm

5.5.2 Transmitter

Table 10: Transmitter Characteristics – Bluetooth LE

Parameter	Conditions	Min	Typ	Max	Unit
RF transmit power	-	-	0	-	dBm
Gain control step	-	-	3	-	dBm
RF power control range	-	–12	-	+9	dBm
Adjacent channel transmit power	F = F0 ± 2 MHz	-	–52	-	dBm
	F = F0 ± 3 MHz	-	–58	-	dBm
	F = F0 ± > 3 MHz	-	–60	-	dBm
$\Delta f1_{\text{avg}}$	-	-	-	265	kHz
$\Delta f2_{\text{max}}$	-	247	-	-	kHz
$\Delta f2_{\text{avg}}/\Delta f1_{\text{avg}}$	-	-	–0.92	-	-
ICFT	-	-	–10	-	kHz
Drift rate	-	-	0.7	-	kHz/50 μs
Drift	-	-	2	-	kHz

6 Schematics

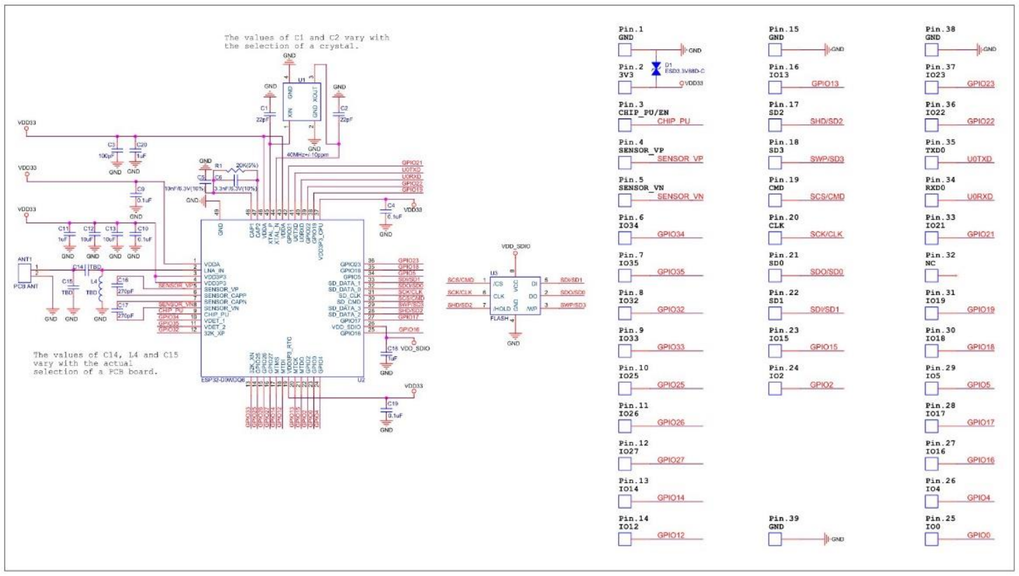
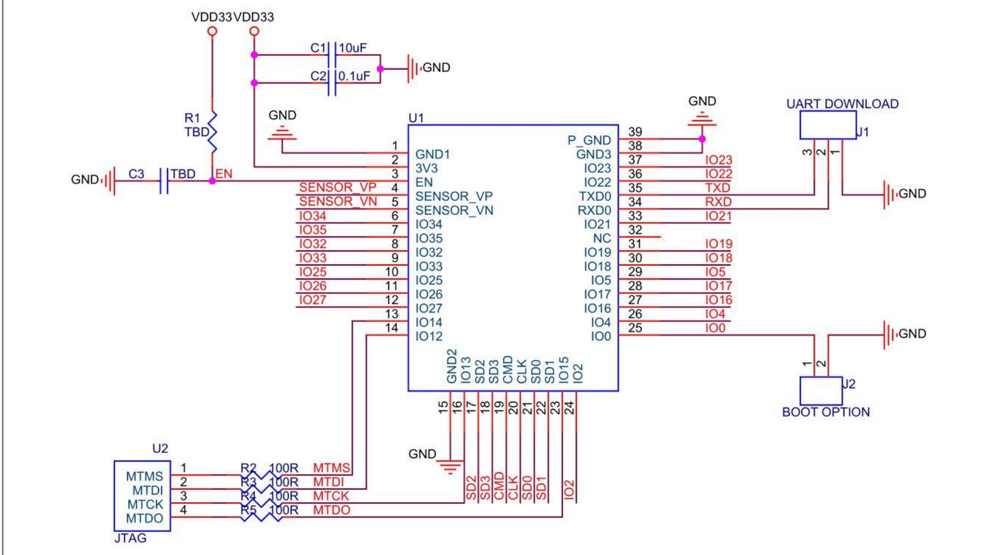


Figure 3: ESP32-WROOM-32 Schematics

7 Peripheral Schematics



MTDI should be kept at a low electric level when powering up the module.

Figure 4: ESP32-WROOM-32 Peripheral Schematics

Note:

- Soldering Pad 39 to the Ground of the base board is not necessary for a satisfactory thermal performance. If users do want to solder it, they need to ensure that the correct quantity of soldering paste is applied.
- To ensure the power supply to the ESP32 chip during power-up, it is advised to add an RC delay circuit at the EN pin. The recommended setting for the RC delay circuit is usually R = 10 kΩ and C = 1 μF. However, specific parameters should be adjusted based on the power-up timing of the module and the power-up and reset sequence timing of the chip. For ESP32's power-up and reset sequence timing diagram, please refer to Section *Power Scheme* in *ESP32 Datasheet*.

8 Physical Dimensions

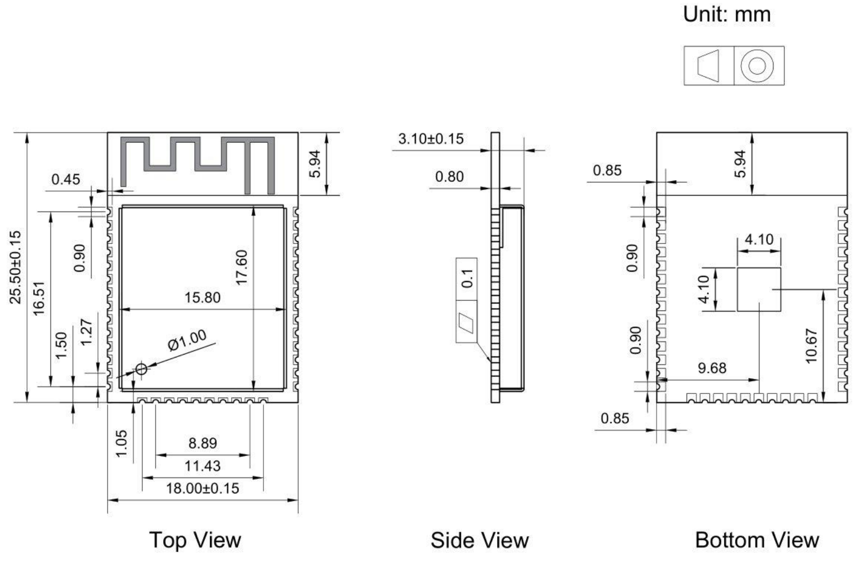


Figure 5: Physical Dimensions of ESP32-WROOM-32

Note:

For information about tape, reel, and product marking, please refer to *Espressif Module Package Information*.

9 Recommended PCB Land Pattern

This section provides the following resources for your reference:

- Figures for recommended PCB land patterns with all the dimensions needed for PCB design. See Figure 6 *Recommended PCB Land Pattern*.
- Source files of recommended PCB land patterns to measure dimensions not covered in Figure 6. You can view the source files for ESP32-WROOM-32 with Autodesk Viewer.

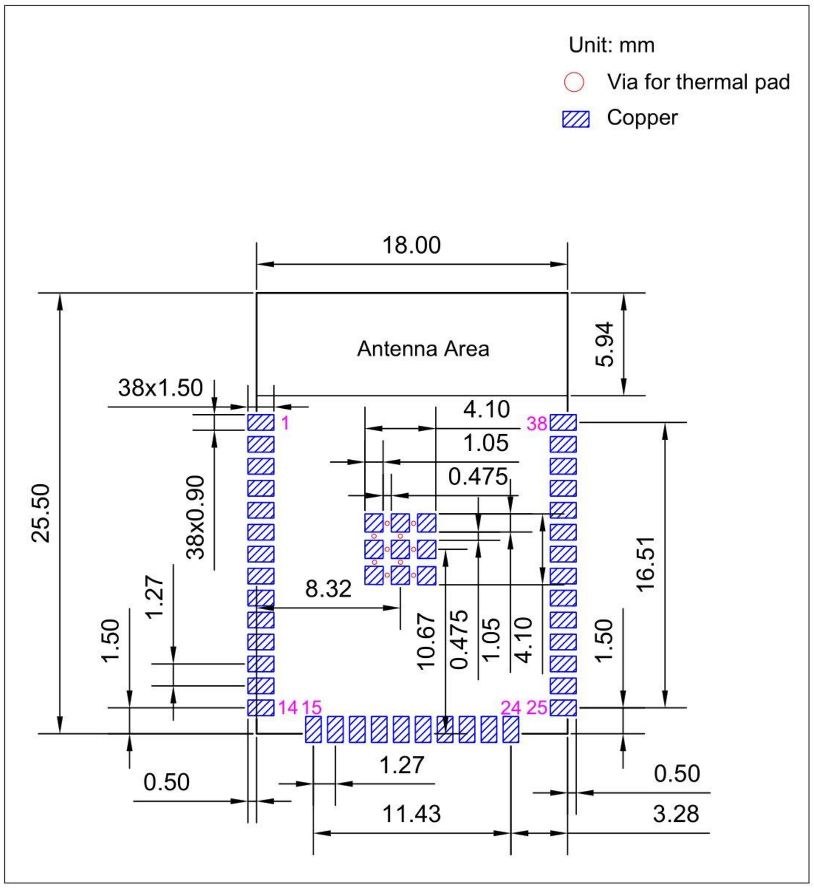


Figure 6: Recommended PCB Land Pattern

10 Product Handling

10.1 Storage Conditions

The products sealed in moisture barrier bags (MBB) should be stored in a non-condensing atmospheric environment of < 40 °C and 90%RH. The module is rated at the moisture sensitivity level (MSL) of 3.

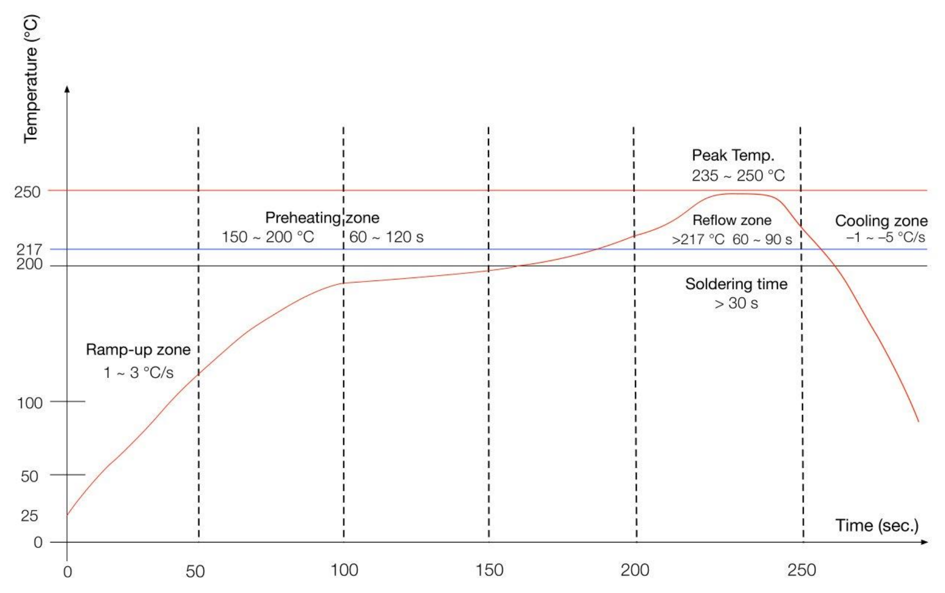
After unpacking, the module must be soldered within 168 hours with the factory conditions 25 ± 5 °C and 60 %RH. If the above conditions are not met, the module needs to be baked.

10.2 Electrostatic Discharge (ESD)

- Human body model (HBM): ±2000 V
- Charged-device model (CDM): ±500 V

10.3 Reflow Profile

Solder the module in a single reflow.



Ramp-up zone — Temp.: 25 ~ 150 °C Time: 60 ~ 90 s Ramp-up rate: 1 ~ 3 °C/s
Preheating zone — Temp.: 150 ~ 200 °C Time: 60 ~ 120 s
Reflow zone — Temp.: >217 °C Time: 60 ~ 90 s; Peak Temp.: 235 ~ 250 °C Time: 30 ~ 70 s
Cooling zone — Peak Temp. ~ 180 °C Ramp-down rate: −1 ~ −5 °C/s
Solder — Sn-Ag-Cu (SAC305) lead-free solder alloy

Figure 7: Reflow Profile

10.4 Ultrasonic Vibration

Avoid exposing Espressif modules to vibration from ultrasonic equipment, such as ultrasonic welders or ultrasonic cleaners. This vibration may induce resonance in the in-module crystal and lead to its malfunction or even failure. As a consequence, **the module may stop working or its performance may deteriorate**.

11 Related Documentation and Resources

Related Documentation

- ESP32 Series Datasheet – Specifications of the ESP32 hardware.
- ESP32 Technical Reference Manual – Detailed information on how to use the ESP32 memory and peripherals.
- ESP32 Hardware Design Guidelines – Guidelines on how to integrate the ESP32 into your hardware product.
- ESP32 ECO and Workarounds for Bugs – Correction of ESP32 design errors.
- *Certificates*
  https://espressif.com/en/support/documents/certificates
- *ESP32 Product/Process Change Notifications (PCN)*
  https://espressif.com/en/support/documents/pcns
- *ESP32 Advisories* – Information on security, bugs, compatibility, component reliability.
  https://espressif.com/en/support/documents/advisories
- *Documentation Updates and Update Notification Subscription*
  https://espressif.com/en/support/download/documents

Developer Zone

- ESP-IDF Programming Guide for ESP32 – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.
  https://github.com/espressif
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.
  https://espressif.com/en/support/download/sdks-demos

Products

- *ESP32 Series SoCs* – Browse through all ESP32 SoCs.
  https://espressif.com/en/products/socs?id=ESP32
- *ESP32 Series Modules* – Browse through all ESP32-based modules.
  https://espressif.com/en/products/modules?id=ESP32
- *ESP32 Series DevKits* – Browse through all ESP32-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector?language=en

Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.
  https://espressif.com/en/contact-us/sales-questions

Revision History

Date	Version	Release notes
2023-02-13	v3.4	Major updates: • Removed contents about hall sensor according to PCN20221202 • Added Section 10: *Product Handling* Other updates: • Added strapping pin timing in Section 2.3: *Strapping Pins* • Added source files of PCB land patterns and 3D models of the modules (if available) in Section 6: *Recommended PCB Land Pattern*
2022.03	v3.3	Added a link to RF certificates in Table 1 Updated Table 5 Added a note below Figure 5 Added Section 11: *Related Documentation and Resources*
2021.08	v3.2	Replaced Espressif Product Ordering Information with ESP Product Selector Updated the description of TWAI in Table 1 Labeled this document as (Not Recommended For New Designs)
2021.02	V3.1	Modified the note below Figure: Reflow Profile. Updated the trade mark from TWAI™ to TWAI® Deleted Reset Circuit and Discharge Circuit for VDD33 Rail in Section 7: *Peripheral Schematics* Updated Figure 5: *Physical Dimensions of ESP32-WROOM-32* and Figure 6: *Recommended PCB Land Pattern*

2020.11	V3.0	Added TWAI^{TM} in Table 1;
		Added a note under Figure: Reflow Profile;
		Updated the C value in RC circuit from 0.1 μF to 1 μF;
		Provided feedback link.

2019.09	V2.9	• Changed the supply voltage range from 2.7 V ~ 3.6 V to 3.0 V ~ 3.6 V; • Added Moisture sensitivity level (MSL) 3 in Table 1 *ESP32-WROOM-32 Specifications*; • Added notes about "Operating frequency range" and "TX power" under Table 8 *Wi-Fi Radio Characteristics*; • Updated Section 7 *Peripheral Schematics* and added a note about RC delay circuit under it; • Updated Figure 6 *Recommended PCB Land Pattern*.
2019.01	V2.8	Changed the RF power control range in Table 10 from –12 ~ +12 to –12 ~ +9 dBm.
2018.10	V2.7	Added "Cumulative IO output current" entry to Table 5: Absolute Maximum Ratings; Added more parameters to Table 7: DC Characteristics.
2018.08	V2.6	• Added reliability test items the module has passed in Table 1: ESP32-WROOM-32 Specifications, and removed software-specific information; • Updated section 3.4: RTC and Low-Power Management; • Changed the module's dimensions from (18±0.2) mm x (25.5 ±0.2) mm x (3.1±0.15) mm to (18.00±0.10) mm x (25.50±0.10) mm x (3.10±0.10) mm; • Updated Figure 8: Physical Dimensions; • Updated Table 8: Wi-Fi Radio.

Date	Version	Release notes
2018.06	V2.5	• Changed the module name to ESP32-WROOM-32; • Deleted Temperature Sensor in Table 1: ESP32-WROOM-32 Specifications; • Updated Chapter 3: Functional Description; • Added Chapter 6: Recommended PCB Land Pattern; Changes to electrical characteristics: • Updated Table 5: Absolute Maximum Ratings; • Added Table 6: Recommended Operating Conditions; • Added Table 7: DC Characteristics; • Updated the values of "Gain control step", "Adjacent channel transmit power" in Table 10: Transmitter Characteristics - BLE.
2018.03	V2.4	Updated Table 1 in Chapter 1.

2018.01	V2.3	Deleted information on LNA pre-amplifier;
		Updated section 3.4 RTC and Low-Power Management;
		Added reset circuit in Chapter 7 and a note to it.

2017.10	V2.2	Updated the description of the chip's system reset in Section 2.3 Strapping Pins; Deleted "Association sleep pattern" in Table "Power Consumption by Power Modes" and added notes to Active sleep and Modem-sleep; Updated the note to Figure 4 Peripheral Schematics; Added discharge circuit for VDD33 rail in Chapter 7 and a note to it.
2017.09	V2.1	Updated operating voltage/power supply range updated to 2.7 ~ 3.6V; Updated Chapter 7.
2017.08	V2.0	Changed the sensitivity of NZIF receiver to -97 dBm in Table 1; Updated the dimensions of the module; Updated Table "Power Consumption by Power Modes" Power Consumption by Power Modes, and added two notes to it; Updated Table 5, 8, 9, 10; Added Chapter 8; Added the link to certification download.

2017.06	V1.9	Added a note to Section 2.1 Pin Layout;
		Updated Section 3.3 Crystal Oscillators;
		Updated Figure 3 ESP-WROOM-32 Schematics;
		Added Documentation Change Notification.

2017.05	V1.8	Updated Figure 1 Top and Side View of ESP32-WROOM-32 (ESP-WROOM-32).
2017.04	V1.7	Added the module's dimensional tolerance; Changed the input impedance value of 50Ω in Table 8 Wi-Fi Radio Characteristics to output impedance value of 30+j10 Ω.
2017.04	V1.6	Added Figure: Reflow Profile.

2017.03	V1.5	Updated Section 2.2 Pin Description;
		Updated Section 3.2 External Flash and SRAM;
		Updated Section 4 Peripherals and Sensors Description.

2017.03	V1.4	Updated Chapter 1 Preface;
		Updated Chapter 2 Pin Definitions;
		Updated Chapter 3 Functional Description;
		Updated Table Recommended Operating Conditions;

Date	Version	Release notes

		Updated Table 8 Wi-Fi Radio Characteristics;
		Updated Section: Reflow Profile;
		Added Chapter Learning Resources.

2016.12	V1.3	Updated Section 2.1 Pin Layout.
2016.11	V1.2	Added Figure 7 Peripheral Schematics.
2016.11	V1.1	Updated Chapter 6 Schematics.
2016.08	V1.0	First release.

Appendix F: E-Ink Display Datasheet

SPECIFICATION

Product Type : EPD Model Number : H3DSHU

Description : Screen Size: 4.2"

Color: Black and White

Display Resolution: 400*300

'DWH : 2017.03.01

:aYHVKDU(OHFWURQLFV

10F, International Science & Technology Building, Fuhong Rd,

Futian District, Shenzhen, China Website: www.ZDYHVKDUH.com

Revision History

Rev.	Issued Date	Revised Contents
1.0	May.05.2015	Preliminary
1.1.	Jul.27.2015	1. In part note 9-2: Modify each update interval time should be minimum at 150 seconds to 180 seconds.

1.2	Aug.21.2015	In part 8: Modify typical operating sequence. In part 12: Delete block diagram.
1.3	Nov.03.2015	1. In part 6: Delete command 70h.
1.4	Nov.11.2015	1. In part 4: Modify the mechanical drawing of EPD module.
2.0	Mar.01.2017	1. In part 7-5): Modify Reference Circuit.

TECHNICAL SPECIFICATION

CONTENTS

NO.	ITEM	PAGE
-	Cover	1
-	Revision History	2
-	Contents	3
1	Application	4
2	Features	4
3	Mechanical Specifications	4
4	Mechanical Drawing of EPD module	5
5	Input/Output Terminals	6

6	Command Table	8
7	Electrical Characteristics	27
8	Typical Operating Sequence	35
9	Optical Characteristics	39
10	Handling, Safety and Environment Requirements	41
11	Reliability test	42
12	Point and line standard	44
13	Packing	45

1. Over View

The display is a TFT active matrix electrophoretic display, with interface and a reference system design. The 4.2” active area contains 400×300 pixels, and has 1-bit white/black full display capabilities. An integrated circuit contains gate buffer, source buffer, interface, timing control logic, oscillator, DC-DC, SRAM, LUT, VCOM, and border are supplied with each panel.

2. Features

High contrast

High reflectance

Ultra wide viewing angle

Ultra low power consumption

Pure reflective mode

Bi-stable

Commercial temperature range

Landscape, portrait mode

Antiglare hard-coated front-surface

Low current deep sleep mode

On chip display RAM

Waveform stored in On-chip OTP

Serial peripheral interface available

On-chip oscillator

On-chip booster and regulator control for generating VCOM, Gate and source driving voltage

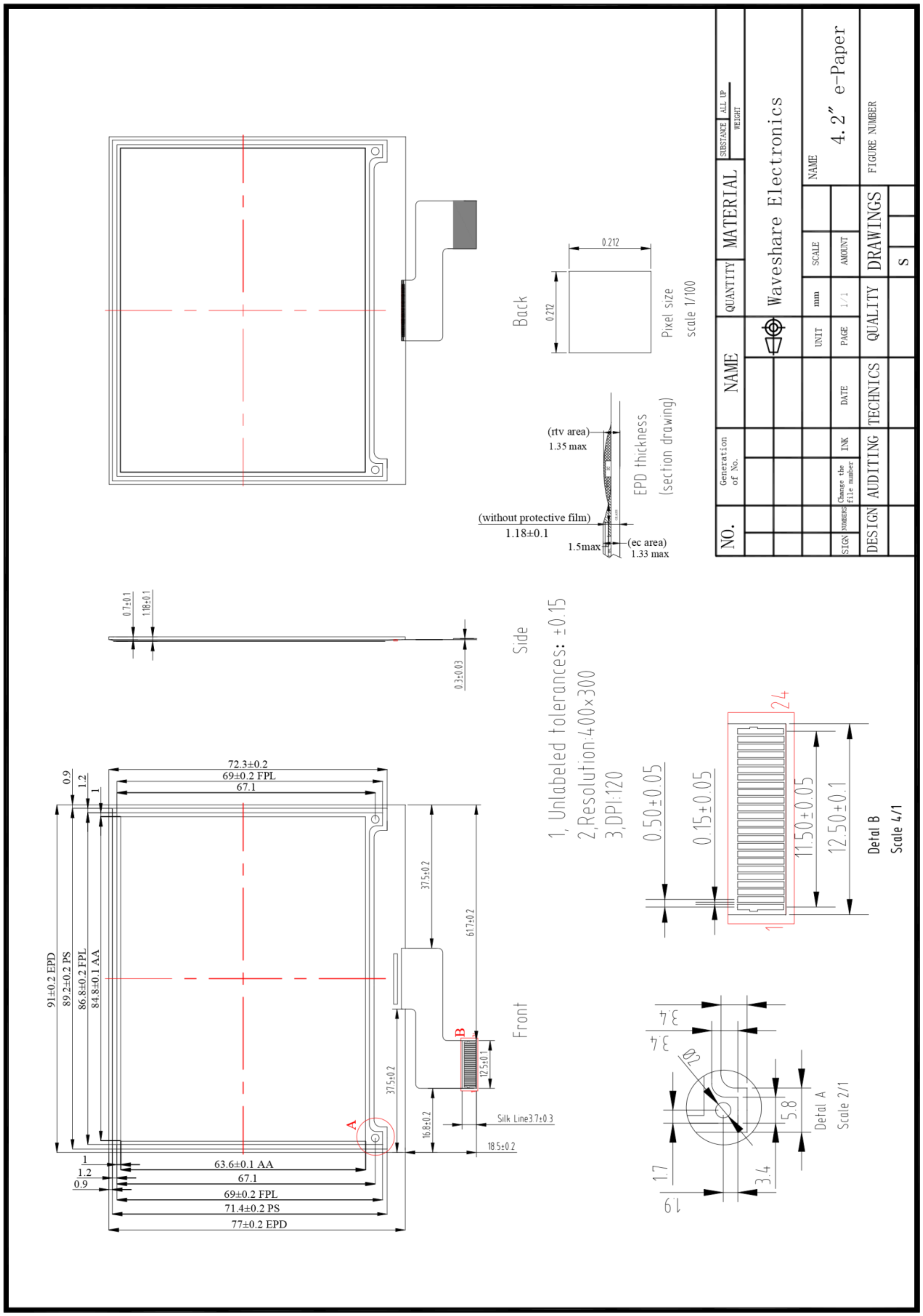
I^2C Signal Master Interface to read external temperature sensor

Available in COG package IC thickness 300um

3. Mechanical Specifications

Parameter	Specifications	Unit	Remark
Screen Size	4.2	Inch	
Display Resolution	400(H)×300(V)	Pixel	Dpi: 120
Active Area	84.8(H)×63.6 (V)	mm	
Pixel Pitch	0.212×0.212	mm	
Pixel Configuration	Square		
Outline Dimension	91.0(H)×77.0(V) ×1.18(D)	mm	
Weight	13.76±0.5	g	

4. Mechanical Drawing of EPD module



Waveshare Electronics
4.2" e-Paper
NO.
NAME
QUANTITY
MATERIAL
SUBSTANCE
ALL UP WEIGHT
Generation of No.
UNIT
mm
SCALE
PAGE
1/1
AMOUNT
DATE
INK
Change the file number
SIGN
NUMBERS
NAME
FIGURE NUMBER
DESIGN
AUDITING
TECHNICS
QUALITY
DRAWINGS
S
Back
0.212
0.212
Pixel size scale 1/100
(rtv area) 1.35 max
(without protective film) 1.18±0.1
1.5max
(ec area) 1.33 max
EPD thickness (section drawing)
0.7±0.1
1.18±0.1
0.3±0.03
Side
1, Unlabeled tolerances: ±0.15
2, Resolution:400×300
3, DPI:120
0.50±0.05
0.15±0.05
11.50±0.05
12.50±0.1
1
24
Detal B
Scale 4/1
72.3±0.2
69±0.2 FPL
67.1
0.9
1.2
1
91±0.2 EPD
89.2±0.2 PS
86.8±0.2 FPL
84.8±0.1 AA
37.5±0.2
61.7±0.2
37.5±0.2
16.8±0.2
12.5±0.1
Silk Line3.7±0.3
18.5±0.2
A
B
63.6±0.1 AA
67.1
69±0.2 FPL
71.4±0.2 PS
77±0.2 EPD
Front
3.4
3.4
Ø2
5.8
1.7
1.9
3.4
Detal A
Scale 2/1

5. Input/Output Terminals

5-1) Pin out List

Pin #	Type	Single	Description	Remark
1		NC	No connection and do not connect with other NC pins	Keep Open
2	O	GDR	N-Channel MOSFET Gate Drive Control	
3	O	RESE	Current Sense Input for the Control Loop	
4	C	VGL	Negative Gate driving voltage	
5	C	VGH	Positive Gate driving voltage	
6	O	TSCL	I^2C Interface to digital temperature sensor Clock pin	
7	I/O	TSDA	I^2C Interface to digital temperature sensor Date pin	
8	I	BS1	Bus selection pin	Note 5-5
9	O	BUSY	Busy state output pin	Note 5-4
10	I	RES #	Reset	Note 5-3
11	I	D/C #	Data /Command control pin	Note 5-2
12	I	CS #	Chip Select input pin	Note 5-1
13	I/O	D0	serial clock pin (SPI)	
14	I/O	D1	serial data pin (SPI)	
15	I	VDDIO	Power for interface logic pins	

16	I	VCI	Power Supply pin for the chip	
17		VSS	Ground	
18	C	VDD	Core logic power pin	
19	C	VPP	Power Supply for OTP Programming	
20	C	VSH	Positive Source driving voltage	
21	C	PREVGH	Power Supply pin for VGH and VSH	
22	C	VSL	Negative Source driving voltage	
23	C	PREVGL	Power Supply pin for VCOM, VGL and VSL	
24	C	VCOM	VCOM driving voltage	

Note 5-1: This pin (CS#) is the chip select input connecting to the MCU. The chip is enabled for MCU communication only when CS# is pulled Low.

Note 5-2: This pin (D/C#) is Data/Command control pin connecting to the MCU. When the pin is pulled HIGH, the data will be

interpreted as data. When the pin is pulled Low, the data will be interpreted as command.

Note 5-3: This pin (RES#) is reset signal input. The Reset is active Low.

Note 5-4: This pin (BUSY) is Busy state output pin. When Busy is low, the operation of chip should not be interrupted and any

commands should not be issued to the module. The driver IC will put Busy pin low when the driver IC is working such as:

Outputting display waveform; or

Programming with OTP

Communicating with digital temperature sensor

Note 5-5: This pin (BS1) is for 3-line SPI or 4-line SPI selection. When it is “Low”, 4-line SPI is selected. When it is “High”, 3-line SPI (9 bits SPI) is selected. Please refer to below Table.

Table: Bus interface selection

BS1	MPU Interface
L	4-lines serial peripheral interface (SPI)
H	3-lines serial peripheral interface (SPI) – 9 bits SPI

6. Command Table

W/R: 0: Write cycle 1: Read cycle C/D: 0: Command 1: Data D7~D0: -: Don’t care #: Valid Data

#	Command	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0	Registers	Default
1	Panel Setting (PSR)	0	0	0	0	0	0	0	0	0	0		00h
		0	1	#	#	#	#	#	#	#	#	RES[1:0],REG,KW/R,UD, SHL,SHD_N,RST_N	0Fh
2	Power Setting (PWR)	0	0	0	0	0	0	0	0	0	1		01h
		0	1	-	-	-	-	-	-	#	#	VDS_EN,VDG_EN	03h
		0	1	-	-	-	-	-	#	#	#	VCOM_HV,VGHL_LV[1:0]	00h
		0	1	-	-	#	#	#	#	#	#	VDH[5:0]	26h
		0	1	-	-	#	#	#	#	#	#	VDL[5:0]	26h
		0	1	-	-	#	#	#	#	#	#	VDHR[5:0]	03h
3	Power OFF(POF)	0	0	0	0	0	0	0	0	1	0		02h
4	Power OFF Sequence Setting(PFS)	0	0	0	0	0	0	0	0	1	1		03h
		0	1	-	-	#	#	-	-	-	-	T_VDS_OF	00h
5	Power ON(PON)	0	0	0	0	0	0	0	1	0	0		04h

6	Power ON Measure(PMES)	0	0	0	0	0	0	0	1	0	1		05h
7	Booster Soft Start(BTST)	0	0	0	0	0	0	0	1	1	0		06h
		0	1	#	#	#	#	#	#	#	#	BT_PHA[7:0]	17h
		0	1	#	#	#	#	#	#	#	#	BT_PHB[7:0]	17h
		0	1	-	-	#	#	#	#	#	#	BT_PHC[5:0]	17h
8	Deep Sleep	0	0	0	0	0	0	0	1	1	1		07h
		0	1	1	0	1	0	0	1	0	1	Check code	A5h
9	Display Start Transmission 1(DTM1, white/black Data) (x-byte command)	0	0	0	0	0	1	0	0	0	0	B/W Pixel Data (400×300)	10h
		0	1	#	#	#	#	#	#	#	#	KPXL[1:8]	00h
		0	1	..	..	..	..	..	..	..	..	..	…
		0	1	#	#	#	#	#	#	#	#	KPXL[n-1:n]	00h
10	Data Stop	0	0	0	0	0	1	0	0	0	1		11h
		1	1	#	-	-	-	-	-	-	-		00h
11	Display Refresh(DRF)	0	0	0	0	0	1	0	0	1	0		12h
12	VCOM LUT(LUTC)	0	0	0	0	1	0	0	0	0	0		20h

	(45-byte command, structure of bytes 2~7 repeated)												

#	Command	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0	Registers	Default
13	W2W LUT (LUTWW) (43-byte command, structure of bytes 2~7 repeated 7 times)	0	0	0	0	1	0	0	0	0	1		21h
14	B2W LUT (LUTBW / LUTR) (43-byte command, structure of bytes 2~7 repeated 7 times)	0	0	0	0	1	0	0	0	1	0		22h
15	W2B LUT (LUTWB / LUTW)	0	0	0	0	1	0	0	0	1	1		23h

	(43-byte command, structure of bytes 2~7 repeated 7 times)												
16	B2B LUT (LUTBB / LUTB) (43-byte command, sturcture of bytes 2~7 repeated 7 times)	0	0	0	0	1	0	0	1	0	0		24h
17	PLL control(PLL)	0	0	0	0	1	1	0	0	0	0		30h
		0	1	-	-	#	#	#	#	#	#	M[2:0],N[2:0]	3Ch
18	Temperature Sensor Calibration (TSC)	0	0	0	1	0	0	0	0	0	0		40h
		1	1	#	#	#	#	#	#	#	#	LM[10:3]/TSR[7:0]	00h
		1	1	#	#	#	-	-	-	-	-	LM[2:0]/-	00h
19	Temperature Sensor Selection (TSE)	0	0	0	1	0	0	0	0	0	1		41h
		0	1	#	-	-	-	#	#	#	#	TSE,TO[3:0]	00h
20	Temperature Sensor Write(TSW)	0	0	0	1	0	0	0	0	1	0		42h
		0	1	#	#	#	#	#	#	#	#	WATTR[7:0]	00h

		0	1	#	#	#	#	#	#	#	#	WMSB[7:0]	00h
		0	1	#	#	#	#	#	#	#	#	WLSB[7:0]	00h
21	Read Temperature Sensor (TSR)	0	0	0	1	0	0	0	0	1	1		43h
		1	1	#	#	#	#	#	#	#	#	RMSB[7:0]	00h
		1	1	#	#	#	#	#	#	#	#	RLSB[7:0]	00h
22	Vcom and data interval setting(CDI)	0	0	0	1	0	1	0	0	0	0		50h
		0	1	#	#	#	#	#	#	#	#	VBD[1:0],DDX[1:0], CDI[3:0]	D7h

#	Command	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0	Registers	Default
23	Lower Power Detection (LPD)	0	0	0	1	0	1	0	0	0	1		51h
		1	1	-	-	-	-	-	-	-	#	LPD	01h
24	TCON setting (TCON)	0	0	0	1	1	0	0	0	0	0		60h
		0	1	#	#	#	#	#	#	#	#	S2G[3:0],G2S[3:0]	22h
25	Resolution setting (TRES)	0	0	0	1	1	0	0	0	0	1		61h
				-	-	-	-	-	-	-	#	HRES[8:3]	00h
		0	1	#	#	#	#	#	0	0	0		00h

		0	1	-	-	-	-	-	-	-	#	VRES[8:0]	00h
		0	1	#	#	#	#	#	#	#	#		00h
26	GSST Setting (GSST)	0	0	0	1	1	0	0	1	0	1		65h
		0	1	-	-	-	-	-	-	-	#	HST[8:3]	00h
		0	1	#	#	#	#	#	0	0	0		00h
		0	1	-	-	-	-	-	-	-	#	VST[8:0]	00h
		0	1	#	#	#	#	#	#	#	#		00h
27	Get Status (FLG)	0	0	0	1	1	1	0	0	0	1		71h
		1	1	-	#	#	#	#	#	#	#	PTL_FLAG,I^{2}C_BUSY,DATA_FLAG,PON,POF,BUSY	02h
28	Auto Measurement Vcom	0	0	1	0	0	0	0	0	0	0		80h
		0	1	-	-	#	#	#	#	#	#	AMVT[1:0],XON,AMVS, AMV,AMVE	10h
29	Read Vcom Value(VV)	0	0	1	0	0	0	0	0	0	1		81h
		1	1	-	-	#	#	#	#	#	#	VV[5:0]	00h
30	VCM_DC Setting (VDCS)	0	0	1	0	0	0	0	0	1	0		82h
		0	1	-	-	#	#	#	#	#	#	VDCS[5:0]	00h

31	Partial Window (PTL)	0	0	1	0	0	1	0	0	0	0		90h
		0	1	-	-	-	-	-	-	-	#	HRST[8:3]	00h
		0	1	#	#	#	#	#	0	0	0		00h
		0	1	-	-	-	-	-	-	-	#	HRED[8:3]	00h
		0	1	#	#	#	#	#	1	1	1		07h
		0	1	-	-	-	-	-	-	-	#	VRST[8:0]	00h
		0	1	#	#	#	#	#	#	#	#		00h
		0	1	-	-	-	-	-	-	-	#	VRED[8:0]	00h
		0	1	#	#	#	#	#	#	#	#		00h
		0	1	-	-	-	-	-	-	-	#	PT_SCAN	01h

#	Command	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0	Registers	Default
32	Partial In (PTIN)	0	0	1	0	0	1	0	0	0	1		91h
33	Partial Out (PTOUT)	0	0	1	0	0	1	0	0	1	0		92h

34	Program Mode (PGM)	0	0	1	0	1	0	0	0	0	0		A0h
		0	1	1	0	1	0	0	1	0	1	Check code = A5h	A5h
35	Active Progrmming (APG)	0	0	1	0	1	0	0	0	0	1		A1h
36	Read OTP (ROTP)	0	0	1	0	1	0	0	0	1	0		A2h
		1	1	-	-	-	-	-	-	-	-	Read Dummy	N/A
		1	1	#	#	#	#	#	#	#	#	Data of Address = 000h	N/A
		1	1	..	..	..	..	..	..	..	..	..	N/A
		1	1	#	#	#	#	#	#	#	#	Data of address = n	N/A
37	Power Saving (PWS)	0	0	1	1	1	0	0	0	1	1		E3h
		0	1	#	#	#	#	#	#	#	#	VCOM_W[3:0],SD_W[3:0]	00h

(1) Panel Setting (PSR) (Register: R00H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Setting the panel	0	0	0	0	0	0	0	0	0	0
	0	1	RES1	RES0	REG_EN	BWR	UD	SHL	SHD_N	RST_N

RES[1:0]: Display Resolution setting (source x gate)

00b: 400x300 (Default) Active source channels: S0 ~ S399. Active gate channels: G0 ~ G299.

01b: 320x300	Active source channels: S0 ~ S319. Active gate channels: G0 ~ G299.
10b: 320x240	Active source channels: S0 ~ S319. Active gate channels: G0 ~ G239.
11b: 200x300	Active source channels: S0 ~ S199. Active gate channels: G0 ~ G299.

REG_EN: LUT selection

0: LUT from OTP. (Default) 1: LUT from register. BWR: Black / White / Red

0: Pixel with B/W/Red. (Default) 1: Pixel with B/W.

UD: Gate Scan Direction

0: Scan down. First line to last line: Gn-1 → Gn-2 → Gn-3 → … → G0

1: Scan up. (default) First line to last line: G0 → G1 → G2 → … → Gn-1

SHL: Source Shift direction

0: Shift left First data to last data: Sn-1 → Sn-2 → Sn-3 → … → S0

1: Shift right. (default) First data to last data: S0 → S1 → S2 → … → Sn-1 SHD_N: Booster Switch

0: Booster OFF, register data are kept, and SEG/BG/VCOM are kept 0V or floating.

1: Booster ON   (Default)

When SHD_N become LOW, charge pump will be turned OFF, register and SRAM data will keep until VDD OFF, and SD output and VCOM will remain previous condition. SHD_N may have two conditions: 0v or floating.

RST_N: Soft Reset

1: No effect   (Default). Booster OFF, Register data are set to their default values, and SEG/BG/VCOM: 0V When RST_N become LOW, the driver will be reset, all registers will be reset to their default value. All driver functions will be disabled. SD output and VCOM will base on previous condition. It may have two conditions: 0v or floating.

(2)   Power Setting (PWR) (R01H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Selecting Internal/External Power	0	0	0	0	0	0	0	0	0	1
	0	1	-	-	-	-	-	-	VDS_EN	VDG_EN
	0	1	-	-	-	-	-	VCOM_HV	VGHL_LV[1:0]	
	0	1	-	-	VDH[5:0]					
	0	1	-	-	VDL[5:0]					
	0	1	-	-	VDHR[5:0]					

VDS_EN: Source power selection

0: External source power from VDH/VDL pins

1: Internal DC/DC function for generating VDH/VDL

VDG_EN: Gate power selection

0: External gate power from VGH/VGL pins

1: Internal DC/DC function for generating VGH/VGL

VCOM_HV: VCOM Voltage Level

0: VCOMH=VDH+VCOMDC, VCOML=VDL+VCOMDC

1: VCOML=VGH, VCOML=VGL

VGHL_LV[1:0]: VGH / VGL Voltage Level selection.

VGHL_LV	VGHL voltage level
00(Default)	VGH=16V,VGL= -16V
01	VGH=15V,VGL= -15V
10	VGH=14V,VGL= -14V
11	VGH=13V,VGL= -13V

VDH[5:0]: Internal VDH power selection for B/W pixel.(Default value: 100110b)

VDH	VDH_V	VDH	VDH_V
000000	2.4V	…	…
000001	2.6V	100110	10.0V
000010	2.8V	100111	10.2V
000011	3.0V	101000	10.4V
000100	3.2V	101001	10.6V
000101	3.4V	101010	10.8V
000110	3.6V	101011	11.0V

000111	3.8V	(others)	11.0V

VDL[5:0]: Internal VDL power selection for B/W pixel. (Default value: 100110b)

VDL	VDL_V	VDL	VDL_V
000000	-2.4V	…	…
000001	-2.6V	100110	-10.0V
000010	-2.8V	100111	-10.2V
000011	-3.0V	101000	-10.4V
000100	-3.2V	101001	-10.6V
000101	-3.4V	101010	-10.8V
000110	-3.6V	101011	-11.0V
000111	-3.8V	(others)	-11.0V

VDHR[5:0]: Internal VDHR power selection for Red pixel. (Default value: 000011b)

VDHR	VDHR _V	VDHR	VDHR _V
000000	2.4V	…	…
000001	2.6V	100110	10.0V
000010	2.8V	100111	10.2V
000011	3.0V	101000	10.4V

000100	3.2V	101001	10.6V
000101	3.4V	101010	10.8V
000110	3.6V	101011	11.0V
000111	3.8V	(others)	11.0V

Power OFF (PWR) (R02H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Turning OFF the power	0	0	0	0	0	0	0	0	1	0

After the Power Off command, the driver will power off following the Power Off Sequence. This command will turn off charge pump, T-con, source driver, gate driver, VCOM, and temperature sensor, but register data will be kept until VDD becomes OFF.

Source Driver output and Vcom will remain as previous condition, which may have 2 condition: 0V or floating.

Power off sequence setting (PFS) (R03H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Setting Power OFF sequence	0	0	0	0	0	0	0	0	1	1
	0	1	-	-	T_VDS_OFF[1:0]		-	-	-	-

T_VDS_OFF[1:0]: Power OFF Sequence of VDH and VDL.

00b: 1frame (Default) 01b: 2 frames 10b: 3frames 11b:4 frame

Power ON (PON) (R04H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0

Turning ON the Power	0	0	0	0	0	0	0	1	0	0

After the Power ON command, the driver will be powered ON following the Power ON Sequence. Refer to the Power ON Sequence section. In the sequence, temperature sensor will be activated for one time sensing before enabling booster.

Power ON Measure (PMES) (R05H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	0	0	0	0	1	0	1

This command enables the internal bandgap, which will be cleared by the next POF.

Booster Soft Start (BTST) (R06H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Starting data transmission	0	0	0	0	0	0	0	1	1	0
	0	1	BT_PHA7	BT_PHA6	BT_PHA5	BT_PHA4	BT_PHA3	BT_PHA2	BT_PHA1	BT_PH 0
	0	1	BT_PHB7	BT_PHB6	BT_PHB5	BT_PHB4	BT_PHB3	BT_PHB2	BT_PHB1	BT_PH 0
	0	1	-	-	BT_PHC5	BT_PHC4	BT_PHC3	BT_PHC2	BT_PHC1	BT_PH 0

BTPHA[7:6]: Soft start period of phase A.

00b: 10mS 01b: 20mS 10b: 30mS 11b: 40mS

BTPHA[5:3]: Driving strength of phase A

000b: strength 1 001b: strength 2 010b: strength 3 011b: strength 4

100b: strength 5 101b: strength 6 110b: strength 7 111b: strength 8 (strongest)

BTPHA[2:0]: Minimum OFF time setting of GDR in phase B

000b: 0.27uS 001b: 0.34uS 010b: 0.40uS 011b: 0.54uS

100b: 0.80uS 101b: 1.54uS 110b: 3.34uS 111b: 6.58uS

BTPHB[7:6]: Soft start period of phase B.

00b: 10mS 01b: 20mS 10b: 30mS 11b: 40mS

BTPHB[5:3]: Driving strength of phase B

000b: strength 1 001b: strength 2 010b: strength 3 011b: strength 4

100b: strength 5 101b: strength 6 110b: strength 7 111b: strength 8 (strongest)

BTPHB[2:0]: Minimum OFF time setting of GDR in phase B

000b: 0.27uS 001b: 0.34uS 010b: 0.40uS 011b: 0.54uS

100b: 0.80uS 101b: 1.54uS 110b: 3.34uS 111b: 6.58uS

BTPHC[5:3]: Driving strength of phase C

000b: strength 1 001b: strength 2 010b: strength 3 011b: strength 4

100b: strength 5 101b: strength 6 110b: strength 7 111b: strength 8 (strongest)

BTPHC[2:0]: Minimum OFF time setting of GDR in phase C

000b: 0.27uS	001b: 0.34uS	010b: 0.40uS	011b: 0.54uS
100b: 0.80uS	101b: 1.54uS	110b: 3.34uS	111b: 6.58uS

Deep Sleep (DSLP) (R07H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Deep Sleep	0	0	0	0	0	0	0	1	1	1
	0	1	1	0	1	0	0	1	0	1

After this command is transmitted, the chip would enter the deep-sleep mode to save power.

The deep sleep mode would return to standby by hardware reset.

The only one parameter is a check code, the command would be executed if check code = 0xA5.

Data Start Transmission 1 (DTM1) (R10H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Starting data transmission	0	0	0	0	0	1	0	0	0	0
	0	1	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8
	0	1	..	..	..	..	..	..	..	..
	0	1	Pixel(n-7)	Pixel(n-6)	Pixel(n-5)	Pixel(n-4)	Pixel(n-3)	Pixel(n-2)	Pixel(n-1)	Pixel(n)

This command starts transmitting data and write them into SRAM. To complete data transmission, command DSP (Data transmission Stop) must be issued. Then the chip will start to send data/VCOM for panel.

In B/W mode, this command writes “OLD” data to SRAM.

In B/W/Red mode, this command writes “B/W” data to SRAM.

In Program mode, this command writes "OTP" data to SRAM for programming.

Data Stop (DSP) (R11H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Stopping data transmission	0	0	0	0	0	1	0	0	0	1
	1	1	Data_flag	-	-	-	-	-	-	-

To stop data transmission, this command must be issued to check the data_flag.

Data_flag: Data flag of receiving user data.

0: Driver didn't receive all the data.

1: Driver has already received all the one-frame data (DTM1 and DTM2).

After "Data Start" (R10h) or "Data Stop" (R11h) commands and when data_flag=1, the refreshing of panel starts and BUSY signal will become "0".

Display Refresh (DRF) (R12H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Refreshing the display	0	0	0	0	0	1	0	0	1	0

While user sent this command, driver will refresh display (data/VCOM) according to SRAM data and LUT.

After Display Refresh command, BUSY signal will become "0" and the refreshing of panel starts.

VCOM LUT (LUTC) (R20H)

This command builds Look-up Table for VCOM

W2W LUT (LUTWW) (R21H)

This command builds Look-up Table for White-to-White.

B2W LUT (LUTBW/LUTR) (R22H)

This command builds Look-up Table for Black-to-White.

W2B LUT (LUTWB/LUTW) (R23H)

This command builds Look-up Table for White - to- Black.

B2B LUT (LUTBB / LUTB) (R24H)

This command builds Look-up Table for Black - to- Black.

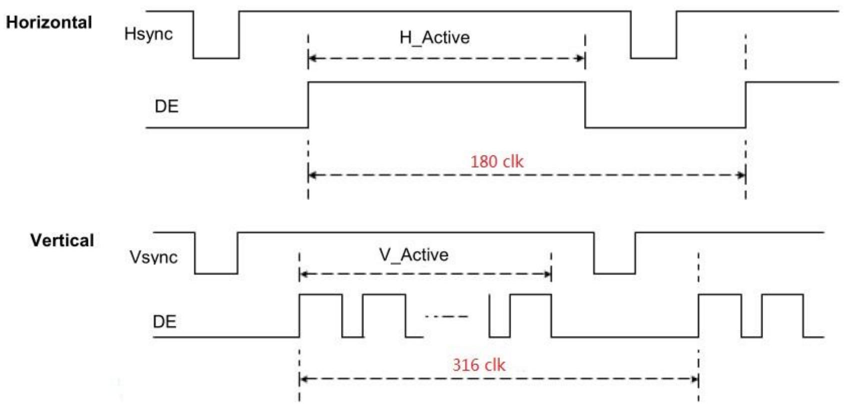
PLL Control (PLL)   (R30H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Controlling PLL	0	0	0	0	1	1	0	0	0	0
	0	1	-	-	M[2:0]			N[2:0]		

The command controls the PLL clock frequency. The PLL structure must support the following frame rates:

M	N	Frame Rate	M	N	Frame Rate	M	N	Frame Rate	M	N	Frame Rate
	1	29 Hz		1	86 Hz		1	150 Hz		1	200 Hz
	2	14 Hz		2	43 Hz		2	72 Hz		2	100 Hz
	3	10 Hz		3	29 Hz		3	48 Hz		3	67 Hz
1	4	7 Hz	3	4	21 Hz	5	4	36 Hz	7	4	50 Hz (Default)
	5	6 Hz		5	17 Hz		5	29 Hz		5	40 Hz
	6	5 Hz		6	14 Hz		6	24 Hz		6	33Hz

	7	4 Hz		7	12Hz		7	20 Hz		7	29 Hz
	1	57 Hz		1	114 Hz		1	171 Hz			
	2	29 Hz		2	57 Hz		2	86 Hz			
	3	19 Hz		3	38 Hz		3	57 Hz			
2	4	14 Hz	4	4	29Hz	6	4	43 Hz			
	5	11 Hz		5	23 Hz		5	34 Hz			
	6	10 Hz		6	19 Hz		6	29 Hz			
	7	8 Hz		7	16 Hz		7	24 Hz			



Temperature Sensor Calibration (TSC) (R40H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Sensing Temperature	0	0	0	1	0	0	0	0	0	0
	1	1	D10/TS7	D9/TS6	D8/TS5	D7/TS4	D6/TS3	D5/TS2	D4/TS1	D3/TS0
	1	1	D2	D1	D0	-	-	-	-	-

This command reads the temperature sensed by the temperature sensor.

TS[7:0]: When TSE (R41h) is set to 0, this command reads internal temperature sensor value.

D[10:0]: When TSE (R41h) is set to 1, this command reads external LM75 temperature sensor value.

TS[7:0]/D[10:3]	Temperature (°C)	TS[7:0]/D[10:3]	Temperature (°C)	TS[7:0]/D[10:3]	Temperature (°C)
1110_0111	-25	0000_0000	0	0001_1001	25
1110_1000	-24	0000_0001	1	0001_1010	26
1110_1001	-23	0000_0010	2	0001_1011	27
1110_1010	-22	0000_0011	3	0001_1100	28
1110_1011	-21	0000_0100	4	0001_1101	29
1110_1100	-20	0000_0101	5	0001_1110	30
1110_1101	-19	0000_0110	6	0001_1111	31
1110_1110	-18	0000_0111	7	0010_0000	32
1110_1111	-17	0000_1000	8	0010_0001	33
1111_0000	-16	0000_1001	9	0010_0010	34
1111_0001	-15	0000_1010	10	0010_0011	35
1111_0010	-14	0000_1011	11	0010_0100	36

1111_0011	-13	0000_1100	12	0010_0101	37
1111_0100	-12	0000_1101	13	0010_0110	38
1111_0101	-11	0000_1110	14	0010_0111	39
1111_0110	-10	0000_1111	15	0010_1000	40
1111_0111	-9	0001_0000	16	0010_1001	41
1111_1000	-8	0001_0001	17	0010_1010	42
1111_1001	-7	0001_0010	18	0010_1011	43
1111_1010	-6	0001_0011	19	0010_1100	44
1111_1011	-5	0001_0100	20	0010_1101	45
1111_1100	-4	0001_0101	21	0010_1110	46
1111_1101	-3	0001_0110	22	0010_1111	47
1111_1110	-2	0001_0111	23	0011_0000	48
1111_1111	-1	0001_1000	24	0011_0001	49

Temperature Sensor Enable (TSE) (R41H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Enable Temperature Sensor/Offset	0	0	0	1	0	0	0	0	0	1
	0	1	TSE	-	-	-	TO[3:0]			

This command selects Internal or External temperature sensor.

TSE: Internal temperature sensor switch

0: Enable (Default) 1: Disable; using external sensor.

TO[3:0]: Temperature offset.

TO[3:0]	Calculation	TO[3:0]	Calculation
0000 b	0	1000	-8
0001	1	1001	-7
0010	2	1010	-6
…	…	…	…
0110	6	1110	-2
0111	7	1111	-1

Temperature Sensor Write (TSW) (R42H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Write External Temperature Sensor	0	0	0	1	0	0	0	0	1	0
	0	1	WATTR[7:0]							
	0	1	WMSB[7:0]							
	0	1	WLSB[7:0]							

This command reads the temperature sensed by the temperature sensor. WATTR: D[7:6]: I^{2}C Write Byte Number

00b : 1 byte (head byte only)

01b : 2 bytes (head byte + pointer)

10b : 3 bytes (head byte + pointer + 1st parameter)

11b : 4 bytes (head byte + pointer + 1st parameter + 2nd parameter)

D[5:3]: User-defined address bits (A2, A1, A0)

D[2:0]: Pointer setting

WMSB[7:0]: MSByte of write-data to external temperature sensor.

WLSB[7:0]: LSByte of write-data to external temperature sensor.

Temperature Sensor Read (TSR) (R43H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Read External Temperature Sensor	0	0	0	1	0	0	0	0	1	1
	1	1	RMSB[7:0]							
	1	1	RLSB[7:0]							

This command reads the temperature sensed by the temperature sensor.

RMSB[7:0]: MSByte read data from external temperature sensor

RLSB[7:0]: LSByte read data from external temperature sensor

VCOM And Data Interval Setting (CDI) (R50H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Set Interval Between Vcom and Data	0	0	0	1	0	1	0	0	0	0
	0	1	VBD[1:0]		DDX[1:0]		CDI[3:0]			

This command indicates the interval of Vcom and data output. When setting the vertical back porch, the total blanking will be kept (20 Hsync).

VBD[1:0]: Border data selection

B/W/Red mode (BWR=0)

DDX[0]	VBD[1:0]	LUT	DDX[0]	VBD[1:0]	LUT
0	00	Floating	1(Default)	00	LUTB
	01	LUTR		01	LUTW
	10	LUTW		10	LUTR
	11	LUTB		11	Floating

B/W mode (BWR=1)

DDX[0]	VBD[1:0]	LUT	DDX[0]	VBD[1:0]	LUT
0	00	Floating	1(Default)	00	Floating
	01	LUTBW (1→0)		01	LUTWB (1→0)
	10	LUTWB (0→1)		10	LUTBW (0→1)
	11	Floating		11	Floating

DDX[1:0]: Data polality.

DDX[1] for RED data, DDX[0] for BW data in the B/W/Red mode.

DDX[0] for B/W mode.

B/W/Red mode (BWR=0)

DDX[1:0]	Data{Red, B/W}	LUT	DDX[1:0]	Data{Red, B/W}	LUT

00	00	LUTW	10	00	LUTR
	01	LUTB		01	LUTR
	10	LUTR		10	LUTW
	11	LUTR		11	LUTB
01(Default)	00	LUTB	11	00	LUTR
	01	LUTW		01	LUTR
	10	LUTR		10	LUTB
	11	LUTR		11	LUTW

B/W mode (BWR=1)

DDX[0]	Data{New, Old}	LUT	DDX[0]	Data{New, Old}	LUT
00	00	LUTWW (0→0)	10	0	LUTBW(1→0)
	01	LUTBW (1→0)		1	LUTWB(0→1)
	10	LUTWB (0→1)	11	0	LUTWB(1→0)
	11	LUTBB (1→1)		1	LUTBW(0→1)

01(Default)	00	LUTBB (0→0)
	01	LUTWB (0→1)
	10	LUTBW (1→0)
	11	LUTWW (1→1)

CDI[3:0]: Vcom and data interval

CDI[3:0]	Vcom and Data Interval	CDI[3:0]	Vcom and Data Interval
0000 b	17 hsync	0110	11
0001	16	0111	10 (Default)
0010	15	…	…
0011	14	1101	4
0100	13	1110	3
0101	12	1111	2

Low Power Detection (LPD) (R51H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	1	0	1	0	0	0	1

Detect Low Power	1	1	-	-	-	-	-	-	-	LPD

This command indicates the input power condition. Host can read this flag to learn the battery condition. LPD: Interval Low Power Detection Flag

0: Low power input (VDD < 2.5V)   1: Normal status (default)

TCON Setting (TCON) (R60H)

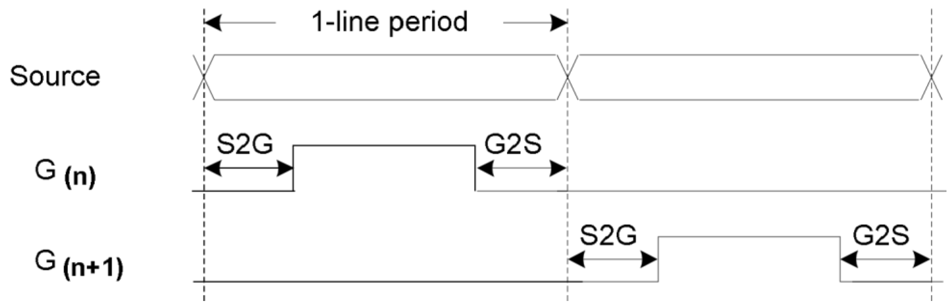
Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Set Gate/Source Non-overlap Period	0	0	0	1	1	0	0	0	0	0
	0	1	S2G[3:0]				G2S[3:0]			

This command defines non-overlap period of Gate and Source.

S2G[3:0] or G2S[3:0]: Source to Gate / Gate to Source Non-overlap period

S2G[3:0] or G2S[3:0]	Period		S2G[3:0] or G2S[3:0]	Period
0000b	4		…	…
0001	8		1011	48
0010	12(Default)		1100	52
0011	16		1101	56
0100	20		1110	60
0101	24		1111	64

Period = 660 nS.



Resolution Setting (TRES) (R61H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Set Display Resolution	0	0	0	1	1	0	0	0	0	1
	0	1								HRES[8]
	0	1	HRES[7:3]					0	0	0
	0	1	-	-	-	-	-	-	-	VRES[8]
	0	1	VRES[7:0]							

This command defines alternative resolution and this setting is of higher priority than the RES[1:0] in R00H (PSR).

HRES[8:3]: Horizontal Display Resolution

VRES[8:0]: Vertical Display Resolution Active channel calculation:

GD : First active gate = G0 (Fixed); LAST active gate = VRES[8:0] - 1

SD : First active source =S0 (Fixed); LAST active source = HRES[8:3]*8 – 1

GSST Setting(GSST) (R65H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Gate/Source	0	0	0	1	1	0	0	1	0	1

Start setting	0	1	-	-	-	-	-	-	-	HST8
	0	1	HST[7:3]					0	0	0
	0	1	-	-	-	-	-	-	-	VST[8]
	0	1	V			ST[7:0]				

This command defines the First Active Gate and First Active Source of active channels.

HST[8:3]: First active source. (Default: S0)

VST[8:0]: First active gate. (Default: G0)

Get Status (FLG) (R71H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Read Flags	0	0	0	1	1	1	0	0	0	1
	1	1	-	PTL_flag	I^2C_ERR	I^2C_ BUSY	data_ flag	PON	POF	BUSY

This command reads the IC status.

PTL_FLAG	Partial display status (high: partial mode)
I^2C_ERR:	I^2C master error status
I^2C_BUSY:	I^2C master busy status (low active)
data_flag:	Driver has already received all the one frame data
PON:	Power ON status
POF:	Power OFF status

BUSY: Driver busy status (low active)

Auto Measure Vcom (AMV) (R80H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Automatically measure Vcom	0	0	1	0	0	0	0	0	0	0
	0	1	-	-	AMVT[1:0]		XON	AMVS	AMV	AMVE

This command reads the IC status.

AMVT[1:0]: Auto Measure Vcom Time

00b: 3s	01b: 5s (Default)
10b: 8s	11b: 10s XON: All Gate ON of AMV

0: Gate normally scan during Auto Measure VCOM period. (default) 1: All Gate ON during Auto Measure VCOM period. AMVS: Source output of AMV

0: Source output 0V during Auto Measure VCOM period. (default) 1: Source output VDHR during Auto Measure VCOM period.

AMV: Analog signal

0: Get Vcom value with the VV command (R81h) (default)

1: Get Vcom value in analog signal. (External analog to digital converter) AMVE: Auto Measure Vcom Enable (/Disable)

0: No effect

1: Trigger auto Vcom sensing.

Vcom Value (VV) (R81H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	1	0	0	0	0	0	0	1

Automatically measure Vcom	1	1	-	-	VV[5:0]

This command gets the Vcom value.

VV[5:0]: Vcom Value Output

VV[5:0]	Vcom value
00 0000b	-0.10 V
00 0001b	-0.15 V
00 0010b	-0.20 V
:	:
11 1010b	-3.00 V

VCM_DC Setting (VDCS) (R82H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Set VCM_DC	0	0	1	0	0	0	0	0	1	0
	0	1	-	-	VDCS[5:0]					

This command sets VCOM_DC value

VDCS[5:0]: VCOM_DC Setting

VDCS[5:0]	Vcom value
00 0000b	-0.10 V (default)
00 0001b	-0.15 V

00 0010b	-0.20 V
:	:
11 1010b	-3.00 V

Partial Window(PTL) (R90H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Set Partial Window	0	0	1	0	0	1	0	0	0	0
	0	1	-	-	-	-	-	-	-	HRST[8]
	0	1	HRST[7:3]					0	0	0
	0	1	-	-	-	-	-	-	-	HRED[8]
	0	1	HRED[7:3]					1	1	1
	0	1	-	-	-	-	-	-	-	VRST[8]
	0	1	VRST[7:0]							
	0	1	-	-	-	-	-	-	-	VRED[8]
	0	1	VRED[7:0]							
	0	1	-	-	-	-	-	-	-	PT_SCAN

This command sets partial window.

HRST[8:3]: Horizontal start channel bank. (value 00h~31h)

HRED[7:3]: Horizontal end channel bank. (value 00h~31h). HRED must be greater than HRST.

VRST[8:0]: Vertical start line. (value 000h~12Bh)

VRED[8:0]: Vertical end line. (value 000h~12Bh). VRED must be greater than VRST.

PT_SCAN: 0: Gates scan only inside of the partial window.

1: Gates scan both inside and outside of the partial window. (default)

Partial In (PTIN) (R91H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Partial In	0	0	1	0	0	1	0	0	0	1

This command makes the display enter
partial mode.

(33) Partial Out (PTOUT) (R92H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Partial In	0	0	1	0	0	1	0	0	1	0

This command makes the display exit partial mode and enter normal mode.

Program Mode (PGM) (RA0H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Enter Program Mode	0	0	1	0	1	0	0	0	0	0
	0	1	1	0	1	0	0	1	0	1

After this command is issued, the chip would enter the program mode.

The mode would return to standby by hardware reset.

The only one parameter is a check code, the command would be excuted if check code = 0xA5.

Active Program (APG) (RA1H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0

Active Program OTP	0	0	1	0	1	0	0	0	0	1

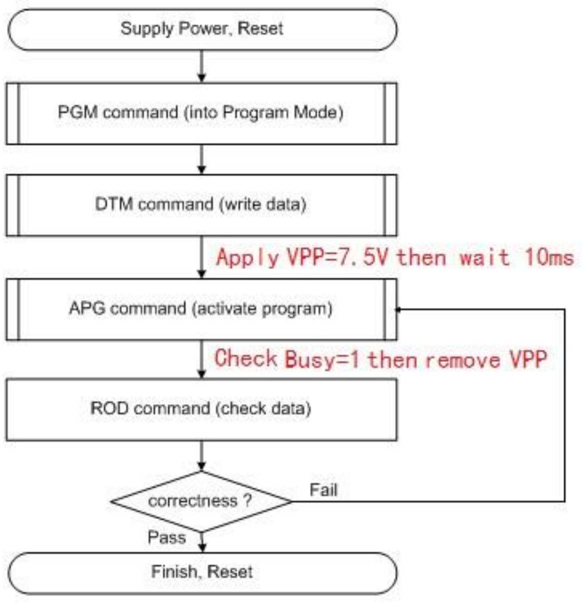
After this command is transmitted, the programming state machine would be activated.

The BUSY flag would fall to 0 until the programming is completed.

Read OTP Data (ROTP)   (RA2H)

Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Read OTP data for check	0	0	1	0	1	0	0	0	1	0
	1	1	Dummy							
	1	1	The data of address 0x000 in the OTP							
	1	1	The data of address 0x001 in the OTP							
	1	1	..							
	1	1	The data of address (n-1) in the OTP							
	1	1	The data of address (n) in the OTP							

The command is used for reading the content of OTP for checking the data of programming. The value of (n) is depending on the amount of programmed data, the max address = 0xFFF.



Supply Power, Reset
PGM command (into Program Mode)
DTM command (write data)
Apply VPP=7.5V then wait 10ms
APG command (activate program)
Check Busy=1 then remove VPP
ROD command (check data)
correctness ?
Fail
Pass
Finish, Reset

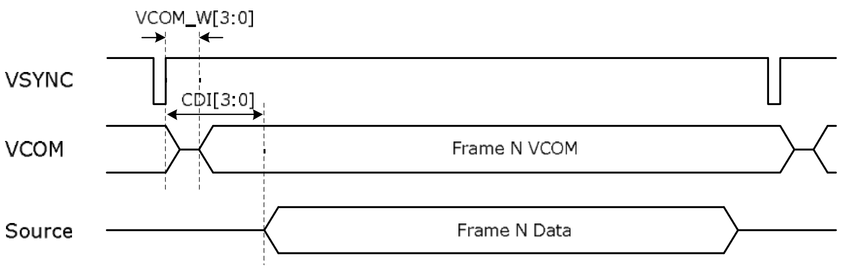
The sequence of programming OTP

Power Saving (PWS) (RE3H)

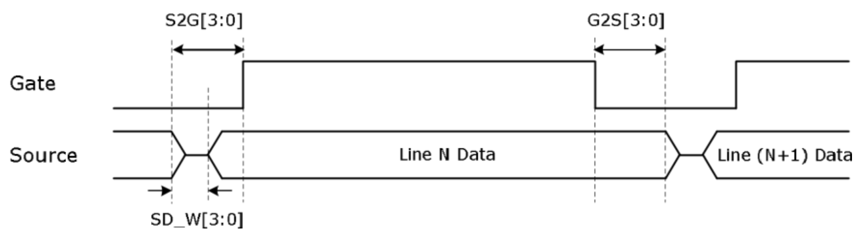
Action	W/R	C/D	D7	D6	D5	D4	D3	D2	D1	D0
Power Saving for Vcom &Source	0	0	1	1	1	0	0	0	1	1
	0	1	VCOM_W[3:0]				SD_W[3:0]			

This command is set for saving power during fresh period. If the output voltage of VCOM / Source is from negative to positive or from positive to negative, the power saving mechanism will be activated. The active period width is defined by the following two parameters.

VCOM_W[3:0]: VCOM power saving width (unit = line period)



SD_W[3:0]: Source power saving width (unit = 660nS)


S2G[3:0]
G2S[3:0]
Gate
Source
Line N Data
Line (N+1) Data
SD_W[3:0]

7. Electrical Characteristics

7-1) Absolute maximum rating

Parameter	Symbol	Rating	Unit
Logic Supply Voltage	V_{CI}	-0.3 to +6.0	V
Logic Input Voltage	V_{IN}	-0.3 to VCI +2.4	V
Operating Temp. range	T_{OPR}	0 to +50	℃
Storage Temp. range	T_{STG}	-25 to +70	℃

7-2) Panel DC Characteristics

The following specifications apply for: VSS = 0V, VCI = 3.3V, TA = 25℃

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Single ground	V_{SS}	-	-	0	-	V
Logic Supply Voltage	VCI	-	2.3	3.3	3.6	V
High level input voltage	VIH	Digital input pins	0.7VCI	-	VCI	V
Low level input voltage	VIL	Digital input pins	0	-	0.3VCI	V
High level output voltage	VOH	Digital input pins , IOH= 400uA	VCI-0.4	-	-	V

Low level output voltage	VOL	Digital input pins , IOL= -400uA	0	-	0.4	V
Image update current	I_{UPDATE}	-	-	8	10	mA
Standby panel current	Istandby	-	-	-	5	uA
Power panel（update）	P_{UPDATE}	-	-	26.4	40	mW
Standby power panel	P_{STBY}	-	-	-	0.0165	mW
Operating temperature	-	-	0	-	50	℃
Storage temperature	-	-	-25	-	70	℃
Image update Time at 25 ℃	-	-	-	6	8	Sec
Deep sleep mode current	I_{VCI}	DC/DC off No clock No input load Ram data not retain	-	2	5	uA
Sleep mode current	I_{VCI}	DC/DC off No clock No input load Ram data retain	-	35	50	uA

- The Typical power consumption is measured with following pattern transition: from horizontal 2 gray scale pattern to vertical

2 gray scale pattern.(Note 7-1)

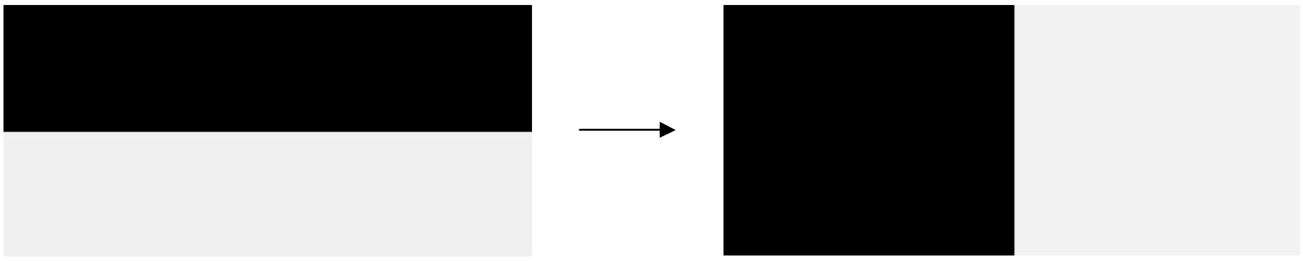
The standby power is the consumed power when the panel controller is in standby mode.

The listed electrical/optical characteristics are only guaranteed under the controller & waveform provided by Waveshare.

Vcom is recommended to be set in the range of assigned value ± 0.1V.

Note 7-1

The Typical power consumption



7-3) Panel AC Characteristics 7-3-1) Oscillator frequency

The following specifications apply for : VSS = 0V, VCI = 3.3V, T_A = 25℃

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Internal Oscillator frequency	Fosc	VCI=2.3 to 3.6V	-	1.625	-	MHz

7-3-2) MCU Interface 7-3-2-1) MCU Interface Selection

In this module, there are 4-wire SPI and 3-wire SPI that can communicate with MCU. The MCU interface mode can be set by hardware selection on BS1 pins. When it is “Low”, 4-wire SPI is selected. When it is “High”, 3-wire SPI (9 bits SPI) is selected.

Pin Name	Data/Command Interface		Control Signal		
Bus interface	D1	D0	CS#	D/C#	RES#

SPI4	SDIN	SCLK	CS#	D/C#	RES#
SPI3	SDIN	SCLK	CS#	L	RES#

Table 7-1: MCU interface assignment under different bus interface mode

Note 7-2: L is connected to VSS

Note 7-3: H is connected to VCI

7-3-2-2) MCU Serial Interface (4-wire SPI)

The 4-wire SPI consists of serial clock SCLK, serial data SDIN, D/C#, CS#. In SPI mode, D0 acts as SCLK, D1 acts as SDIN.

Function	CS#	D/C#	SCLK
Write Command	L	L	↑
Write data	L	H	↑

Table 7-2: Control pins of 4-wire Serial Peripheral interface

Note 7-4: ↑stands for rising edge of signal

SDIN is shifted into an 8-bit shift register in the order of D7, D6, ... D0. The data byte in the shift register is written to the Graphic Display Data RAM (RAM) or command register in the same clock. Under serial mode, only write operations are allowed.

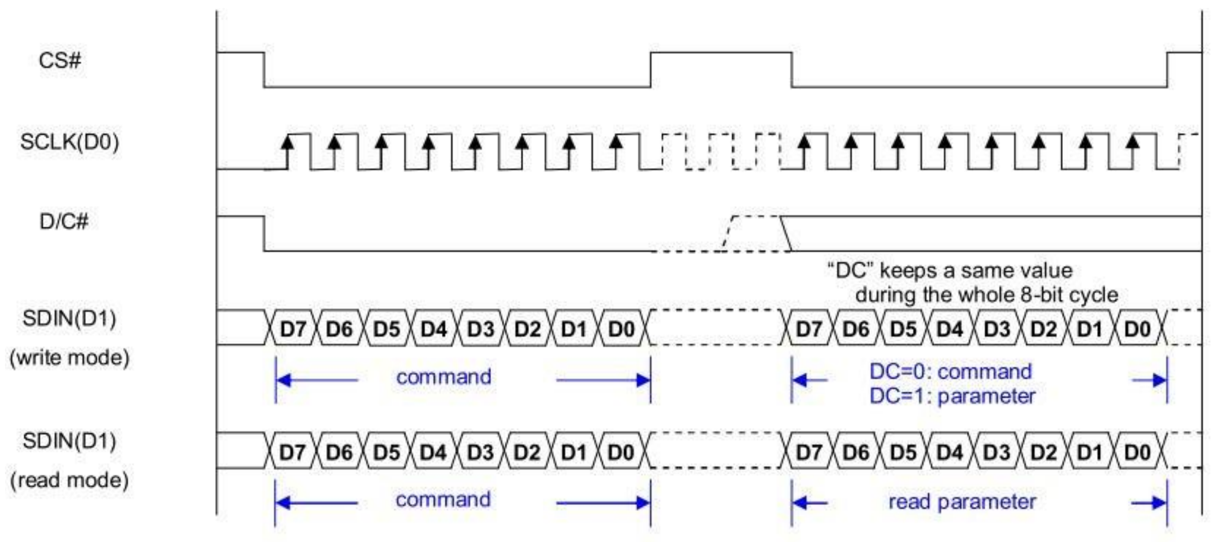


Figure 7-1: Write procedure in 4-wire Serial Peripheral Interface mode

7-3-2-3) MCU Serial Interface (3-wire SPI)

The 3-wire serial interface consists of serial clock SCLK, serial data ADIN and CS#.

In 3-wire SPI mode, D0 acts as SCLK, D1 acts as SDIN, The pin D/C# can be connected to an external ground.

The operation is similar to 4-wire serial interface while D/C# pin is not used. There are altogether 9-bits will be shifted into the shift register on every ninth clock in sequence: D/C# bit, D7 to D0 bit. The D/C# bit (first bit of the sequential data) will determine the following data byte in shift register is written to the Display Data RAM (D/C# bit = 1) or the command register (D/C# bit = 0).Under serial mode, only write operations are allowed.

Function	CS#	D/C#	SCLK
Write Command	L	Tie LOW	↑
Write data	L	Tie LOW	↑

Table 7-3: Control pins of 3-wire Serial Peripheral Interface

Note 7-5: ↑stands for rising edge of signal

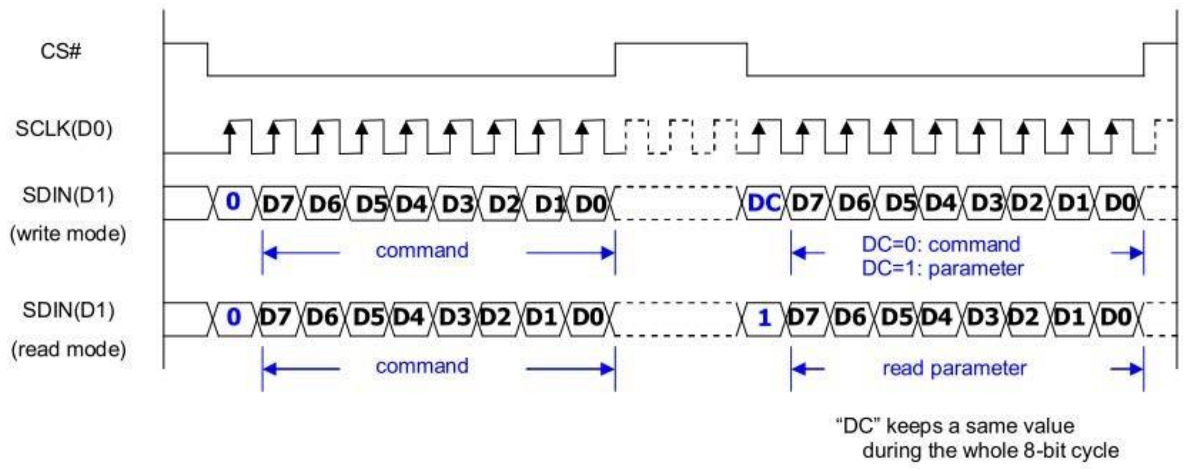
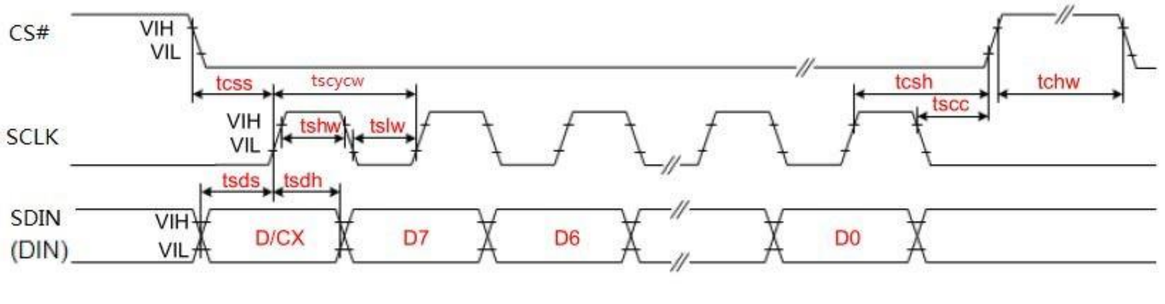
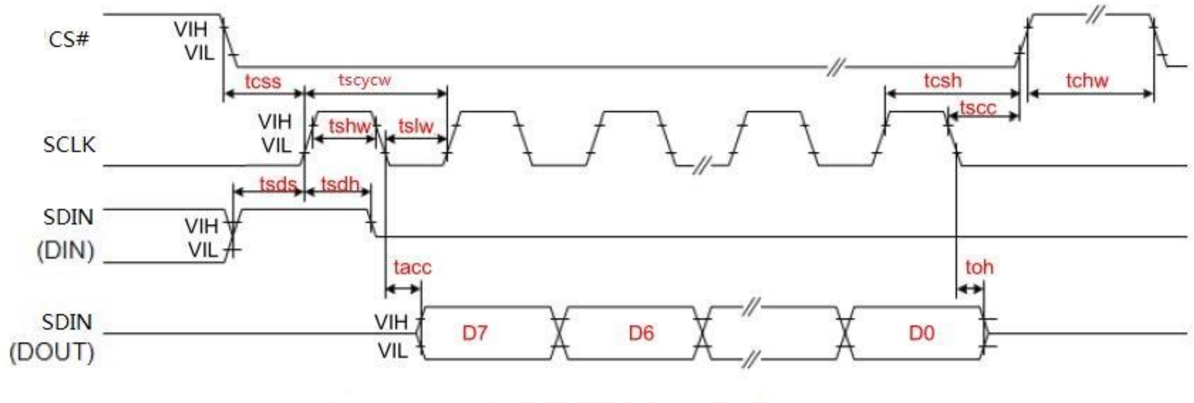


Figure 7-2: Write procedure in 3-wire Serial Peripheral Interface mode

7-3-3) Timing Characteristics of Series Interface



3-wire Serial Interface – Write



3-wire Serial Interface – Read

Symbol	Signal	Parameter	Min	Typ	Max	Unit
tcss	CS#	Chip Select Setup Time	60	-	-	ns
tcsh		Chip Select Hold Time	65	-	-	ns

tscc		Chip Select Setup Time	20	-	-	ns
tchw		Chip Select Setup Time	40	-	-	ns
tscycw		Serial clock cycle (write)	100	-	-	ns
tshw		SCL "H" pulse width (write)	35	-	-	ns
tslw		SCL"L" pulse width (write)	35	-	-	ns
tscycr	SCLK	Serial clock cycle (Read)	150	-	-	ns
tshr		SCL "H" pulse width (Read)	60	-	-	ns
tslr		SCL "L" pulse width (Read)	60	-	-	ns
tsds		Data setup time	30	-	-	ns
tsdh	SDIN (DIN) (DOUT)	Data hold time	30	-	-	ns
tacc		Access time	-	-	10	ns
toh		Output disable time	15	-	-	ns

7-4) Power Consumption

Parameter	Symbol	Conditions	TYP	Max	Unit	Remark

Panel power consumption during update	-	25℃	26.4	40	mW	-
Power consumption in standby mode	-	25℃	-	0.0165	mW	-

7-5) Reference Circuit

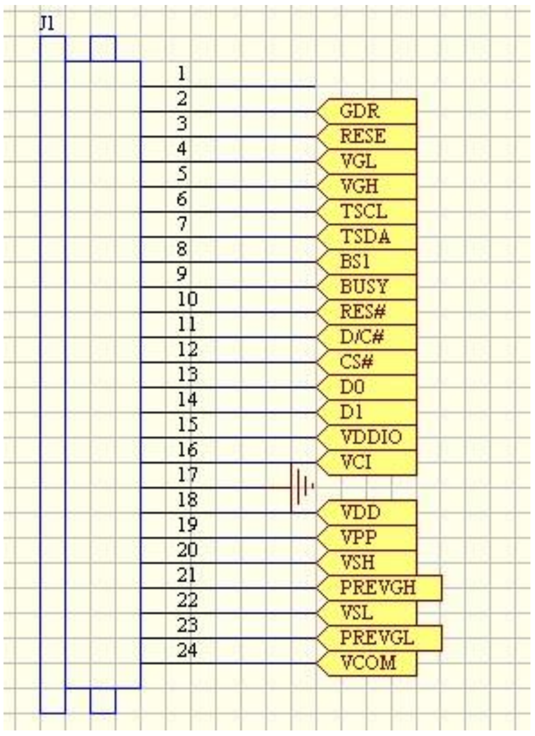


Figure . 7-5 (1)

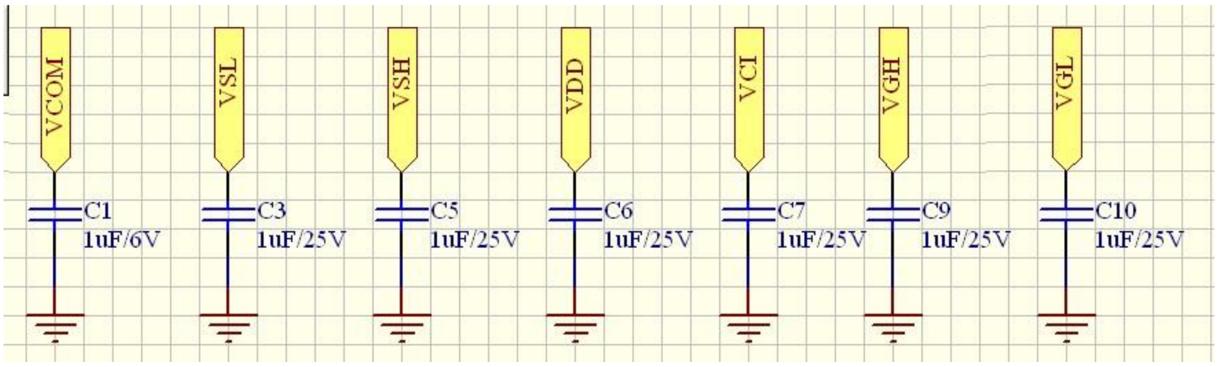


Figure . 7-5 (2)

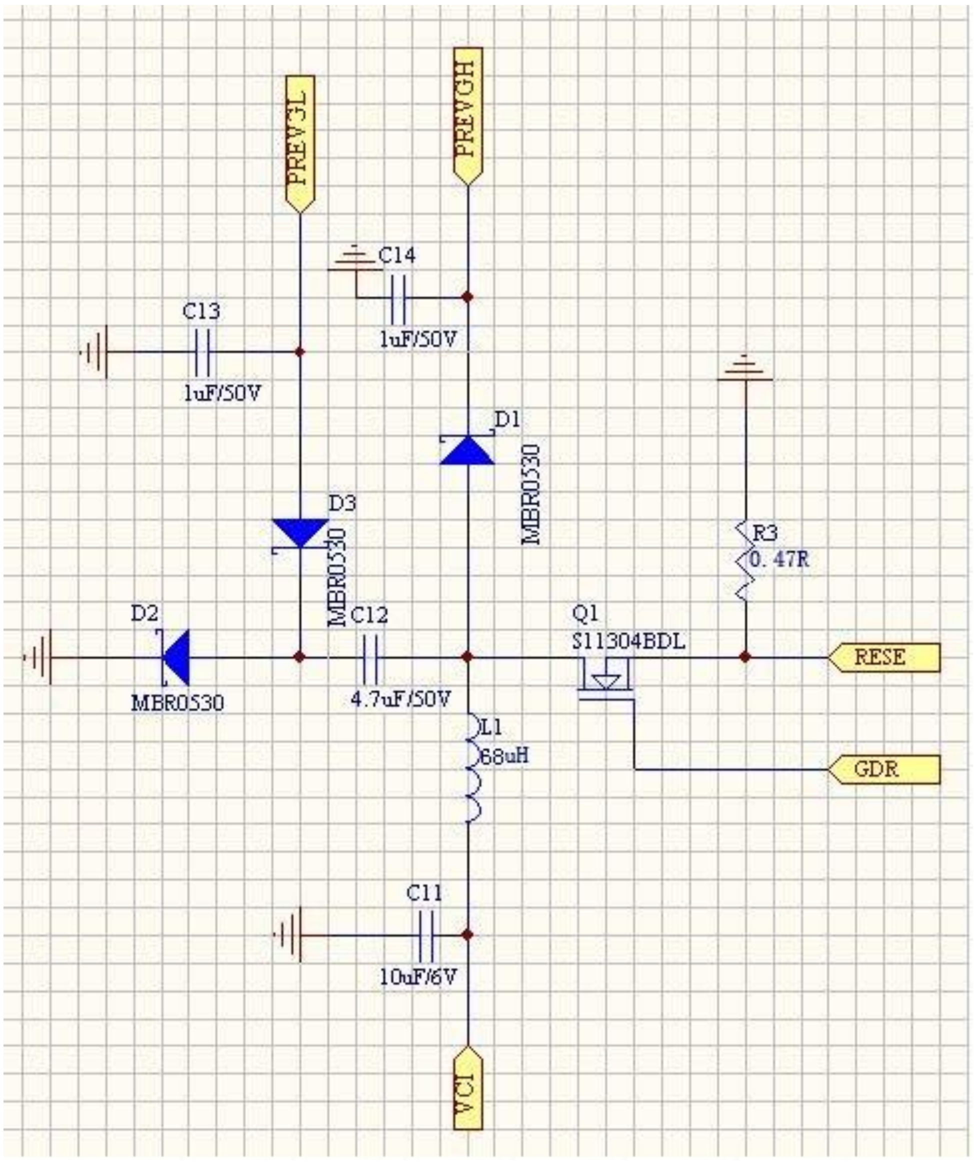

PREVGL
PREVGH
C14
1uF/50V
C13
1uF/50V
D1
MBR0530
D3
MBR0530
R3
0.47R
D2
MBR0530
C12
4.7uF/50V
Q1
SI1304BDL
RESE
L1
68uH
GDR
C11
10uF/6V
VCI

Figure . 7-5 (3)

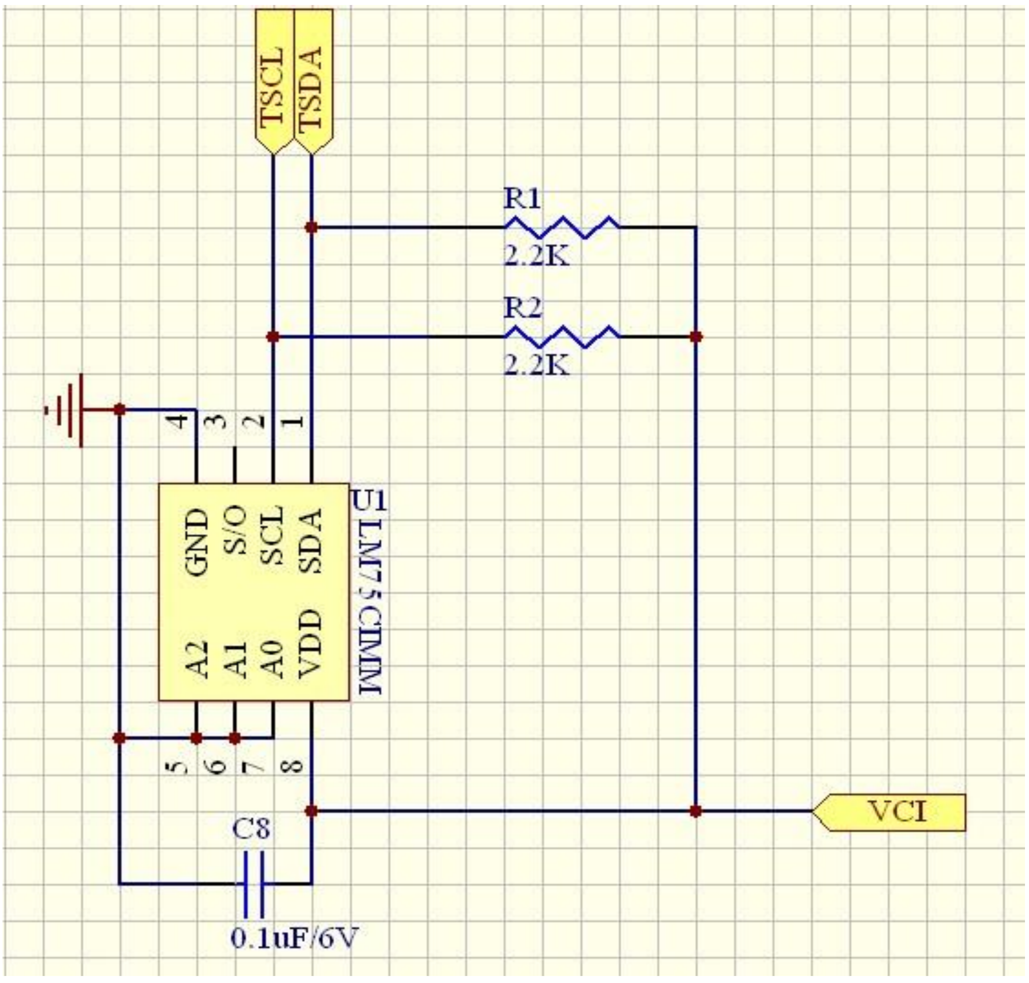
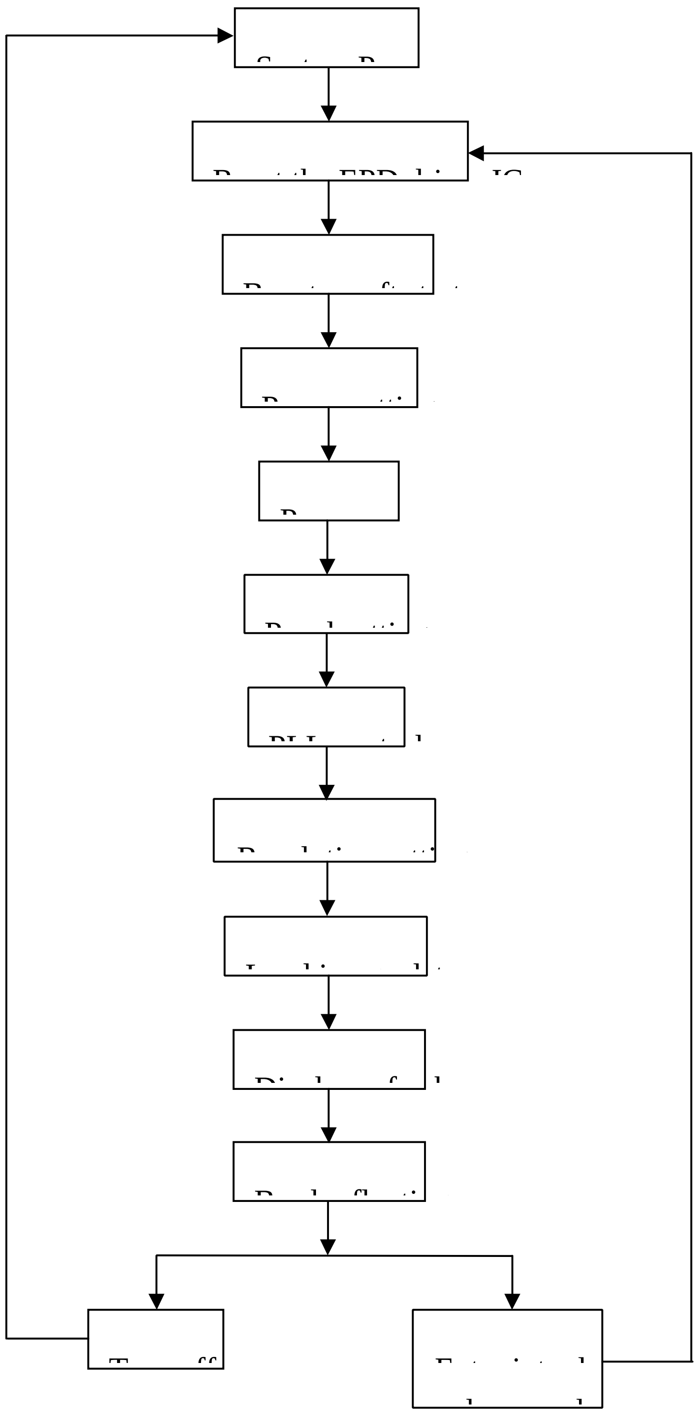


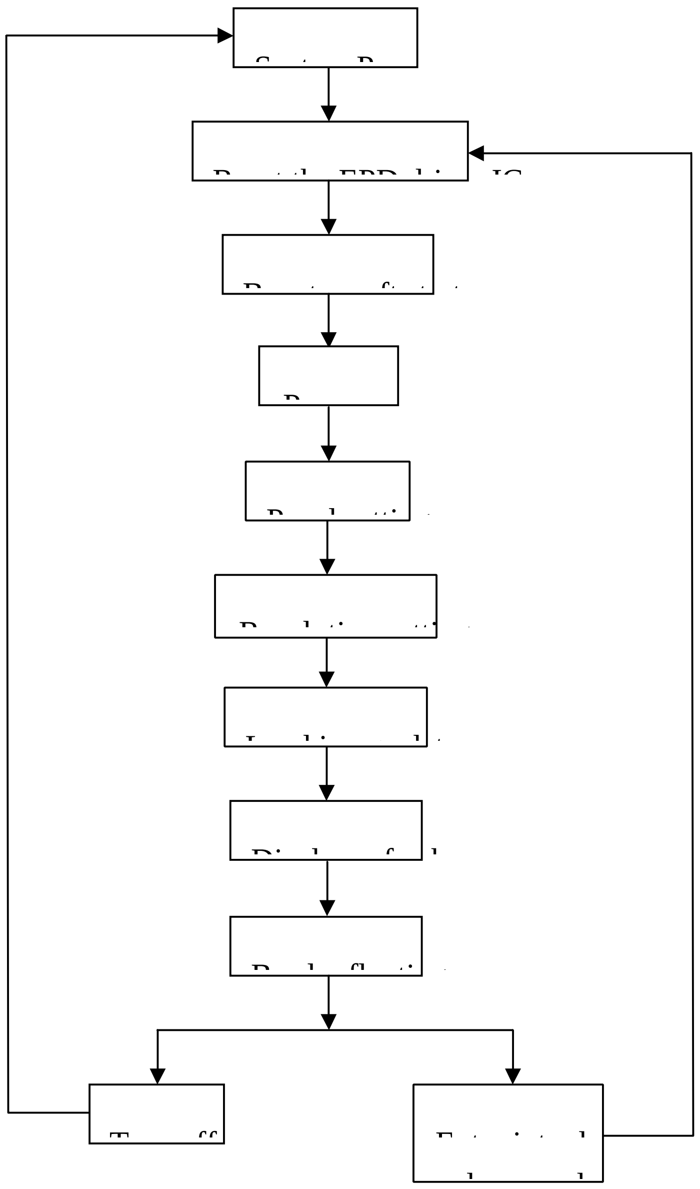
Figure . 7-5 (4)

8. Typical Operating Sequence 8-1) Normal Operation Flow

BWR mode & LUT form Register

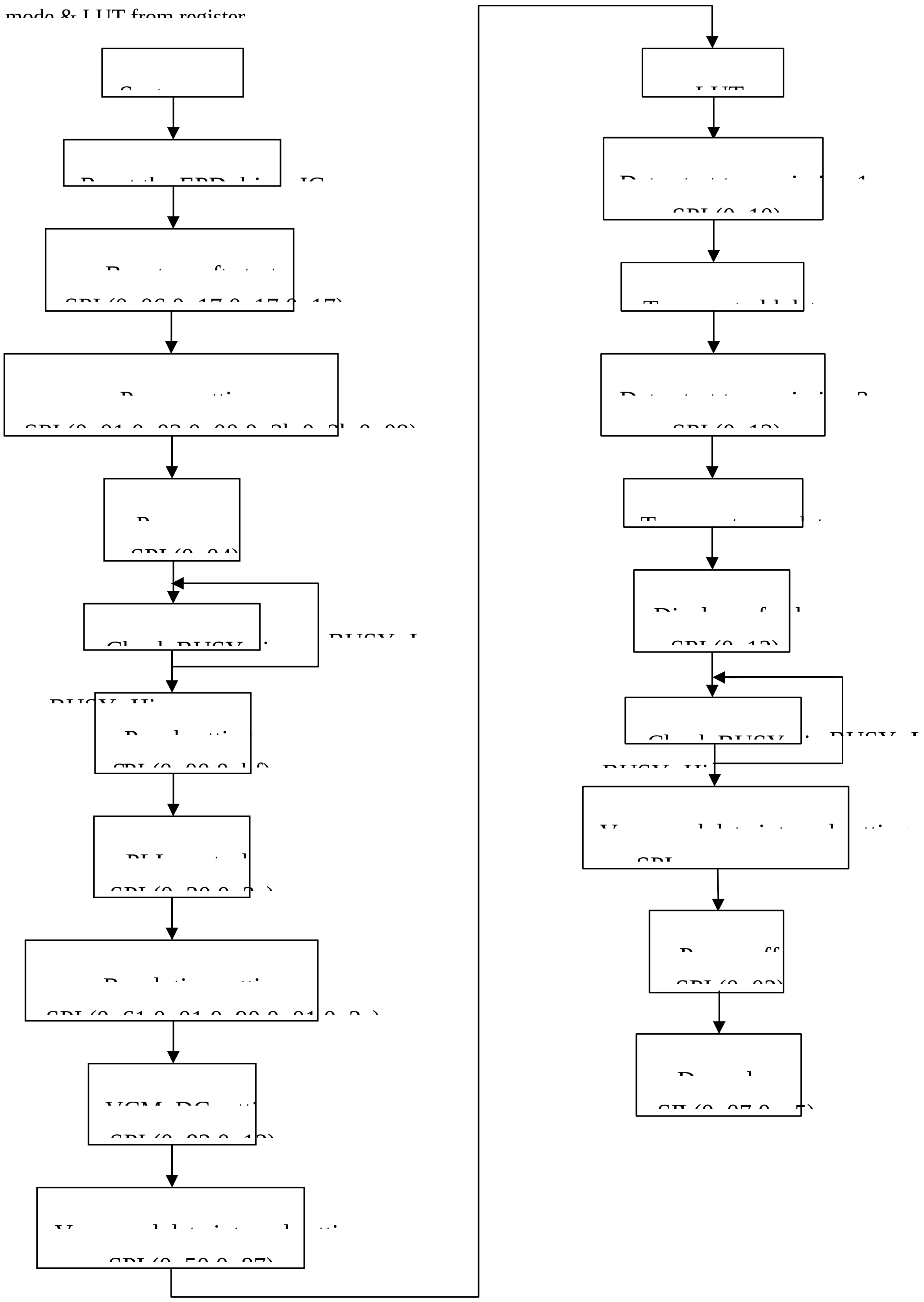


BWR mode & LUT form OTP



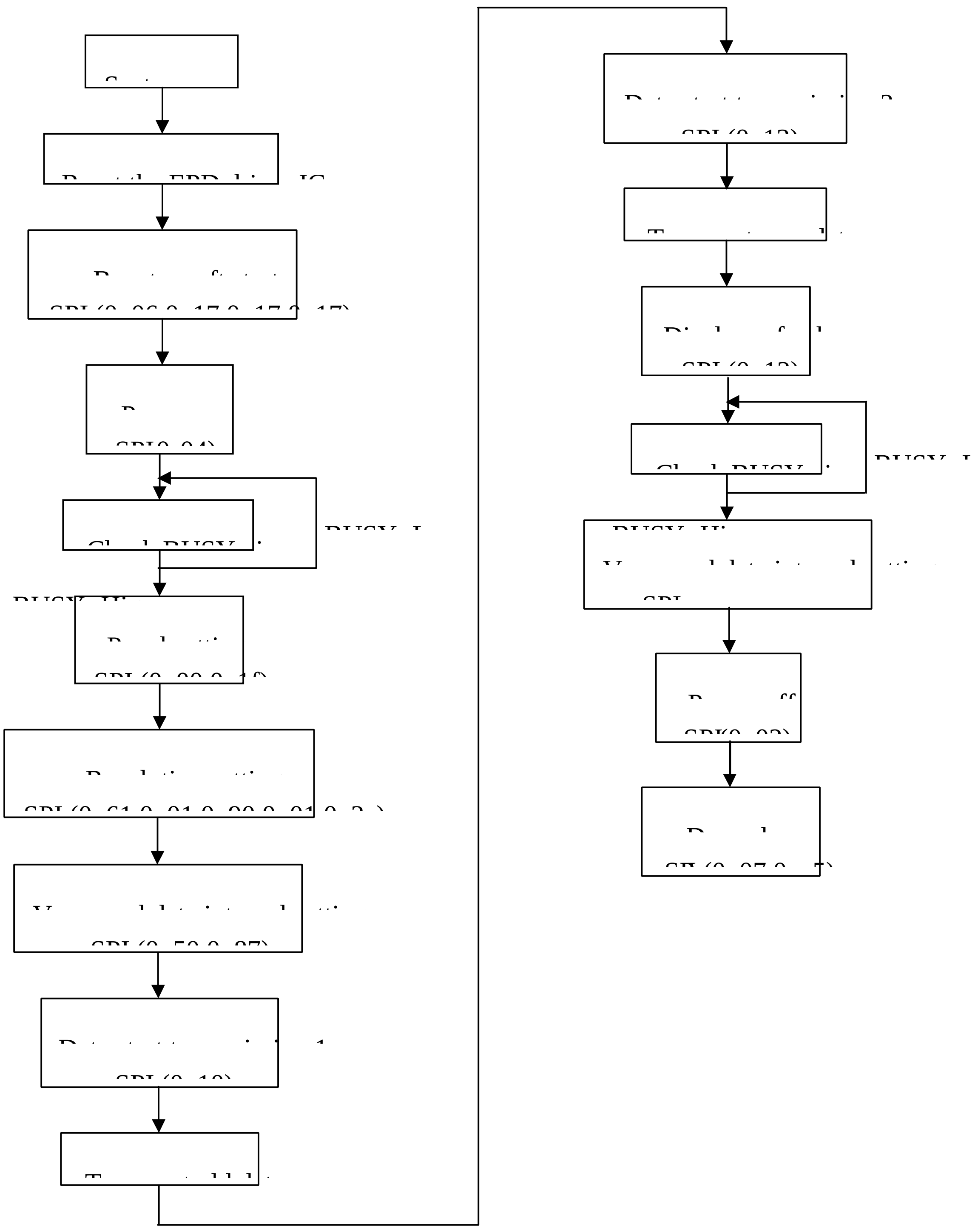
8-2) Reference Program Code

1.BWR mode & LUT from register



Note1: Set border to floating.

2. BWR mode & LUT from OTP



Note1: Set border to floating.

9. Optical characteristics

9-1) Specifications

Measurements are made with that the illumination is under an angle of 45 degrees, the detection is perpendicular unless otherwise specified.

T=25℃

SYMBOL	PARAMETER	CONDITIONS	MIN	TYPE	MAX	UNIT	Note
R	Reflectance	White	30	35	-	%	Note 9-1
Gn	2Grey Level	-	-	DS＋(WS-DS)×n(m-1)	-	L*	-
CR	Contrast Ratio	indoor	8		-	-	-
Panel's life		0℃~50℃		1000000 times or 5 years			Note 9-2

WS: White state, DS: Dark state Gray state from Dark to White : DS、WS m: 2

Note 9-1: Luminance meter: Eye – One Pro Spectrophotometer

Note 9-2: Panel life will not guaranteed when work in temperature below 0 degree or above 50 degree. Each update interval time should be minimum at 180 seconds.

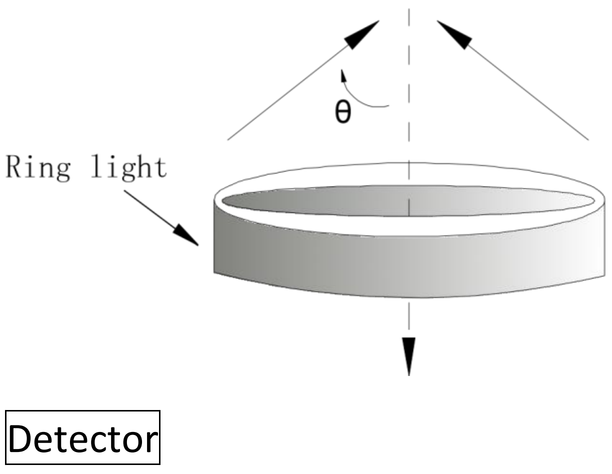
9-2) Definition of contrast ratio

The contrast ratio (CR) is the ratio between the reflectance in a full white area (R1) and the reflectance in a dark area (Rd)() :

R1: white reflectance   Rd: dark reflectance

CR = R1/Rd

Display

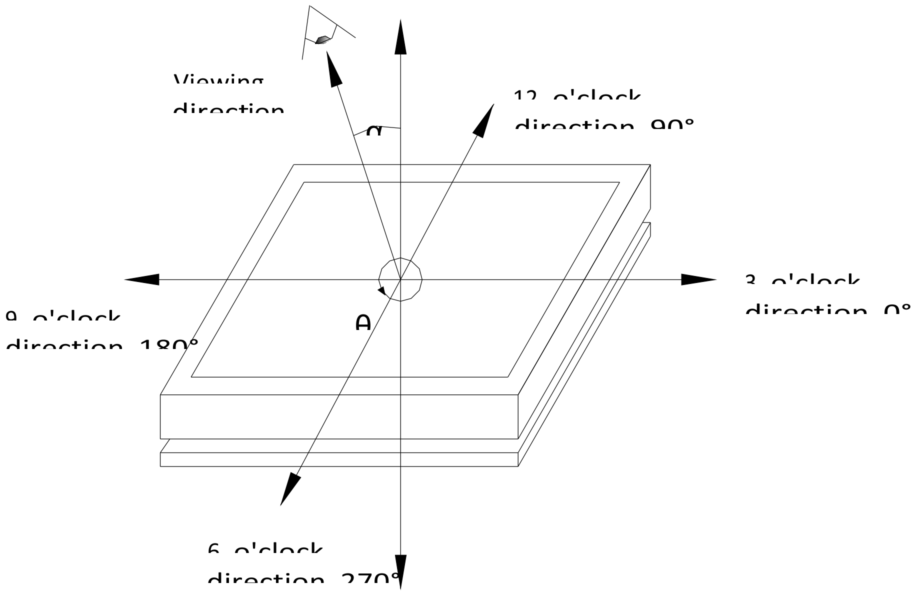


9-3) Reflection Ratio

The reflection ratio is expressed as:

R = Reflectance Factor $_{\text{white board}}$ x (L $_{\text{center}}$ / L $_{\text{white board}}$)

L $_{\text{center}}$ is the luminance measured at center in a white area (R=G =B=1). L $_{\text{white board}}$ is the luminance of a standard white board. Both are measured with equivalent illumination source. The viewing angle shall be no more than 2 degrees.



9-4) Bi-stability

The Bi-stability standard as follows:

Bi-stability	Result

24 hours Luminance drift		AVG	MAX
	White state △L*	-	3
	Black state △L*	-	3

10. Handling, Safety and Environmental Requirements

Warning
The display glass may break when it is dropped or bumped on a hard surface. Handle with care. Should the display break, do not touch the electrophoretic material. In case of contact with electrophoretic material, wash with water and soap.
Caution
The display module should not be exposed to harmful gases, such as acid and alkali gases, which corrode electronic components.
Disassembling the display module can cause permanent damage and invalidate the warranty agreements.

Observe general precautions that are common to handling delicate electronic components. The glass can break and front surfaces can easily be damaged. Moreover the display is sensitive to static electricity and other rough environmental conditions.

Data sheet status	
Product specification	The data sheet contains final product specifications.
Limiting values	

Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.
Application information
Where application information is given, it is advisory and dose not form part of the specification.
Product environmental certification
RoHS

11. Reliability test

	TEST	CONDITION	METHOD	REMARK
1	High-Temperatu re Operation	T = 50℃, RH=35% for 240 hrs	When the experimental cycle finished, the EPD samples will be taken out from the high temperature environmental chamber and set aside for a few minutes. As EPDs return to room temperature, testers will observe the appearance, and test electrical and optical performance based on standard # IEC 60 068-2-2Bp.	When experiment finished, the EPD must meet electrical and optical performance standards.
2	Low-Temperatu re Operation	T = 0℃ for 240 hrs	When the experimental cycle finished, the EPD samples will be taken out from the low temperature environmental chamber and set aside for a few minutes. As EPDs return	When experiment finished, the EPD

			room temperature, testers will observe the appearance, and test electrical and optical performance based on standard # IEC 60 068-2-2Ab.	must meet electrical and optical performance standards.
3	High-Temperatu re Storage	T = +70℃, RH=35% for 240 hrs Test in white pattern	When the experimental cycle finished, the EPD samples will be taken out from the high temperature environmental chamber and set aside for a few minutes. As EPDs return to room temperature, testers will observe the appearance, and test electrical and optical performance based on standard # IEC 60 068-2-2Bp.	When experiment finished, the EPD must meet electrical and optical performance standards.
4	Low-Temperatu re Storage	T = -25℃ for 240 hrs Test in white pattern	When the experimental cycle finished, the EPD samples will be taken out from the low temperature environmental chamber and set aside for a few minutes. As EPDs return to room temperature, testers will observe the appearance, and test electrical and optical performance based on standard # IEC 60 068-2-2Ab	When experiment finished, the EPD must meet electrical and optical performance standards.
5	High Temperature, High-Humidity	T=+40℃, RH=80% for240hrs	When the experimental cycle finished, the EPD samples will be taken out from the environmental chamber and set aside for a few minutes. As EPDs return to room temperature, testers will observe the	When experiment finished, the EPD

	Operation		appearance, and test electrical and optical performance based on standard # IEC 60 068-2-3CA.	must meet electrical and optical performance standards.
6	High Temperature, High-Humidity Storage	T=+60℃, RH=80% for240hrs Test in white pattern	When the experimental cycle finished, the EPD samples will be taken out from the environmental chamber and set aside for a few minutes. As EPDs return to room temperature, testers will observe the appearance, and test electrical and optical performance based on standard # IEC 60 068-2-3CA.	When experiment finished, the EPD must meet electrical performance standards.
7	Temperature Cycle	[-25℃ 30mins]→ [+70℃, RH=35% 30mins],	1. Samples are put in the Temp & Humid. Environmental Chamber. Temperature cycle starts with -25℃, storage period 30 minutes. After 30 minutes, it needs 30min to	When experiment finished, the EPD must meet electrical

		70cycles, Test in white pattern	let temperature rise to 70°C. After 30min, temperature will be adjusted to 70°C and storage period is 30 minutes. After 30 minutes, it needs 30min to let temperature rise to -25°C. One temperature cycle (2hrs) is complete. Temperature cycle repeats 70 times. When 70 cycles finished, the samples will be taken out from experiment chamber and set aside a few minutes. As EPDs return to room temperature, tests will observe the appearance, and test electrical and optical performance 2. based on standard # IEC 60 068-2- 3. 14NB.	and optical performance standards.
8	UV exposure Resistance	765 W/m^2 for 168 hrs,40°C	Standard # IEC 60 068-2-5 Sa	
9	Electrostatic discharge	Machine model: +/-250V, 0Ω,200pF	Standard # IEC61000-4-2	
10	Package Vibration	1.04G,Frequency : 10~500Hz Direction : X,Y,Z	Full packed for shipment	

		Duration:1hours in each direction		
11	Package Drop Impact	Drop from height of 122 cm on Concrete surface Drop sequence:1 corner, 3edges, 6face One drop for each.	Full packed for shipment	

Actual EMC level to be measured on customer application.

Note: (1) The protective film must be removed before temperature test.

(2) In order to make sure the display module can provide the best display quality, the update should be made after putting the display module in stable temperature environment for 15 mins.

12. Point and line standard

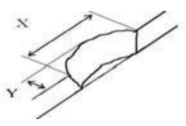
Shipment Inseption Standard

Part-A：Active area   Part-B：Border area

Equipment：Electrical test fixture, Point gauge

Outline dimension：

91.0(H)×77.0(V) ×1.18(D)   Unit：mm

Environment	Temperature	Humidity	Illuminance	Distance	Time	Angle
	23±2℃	55±5%RH	1200～1500Lux	300 mm	35 Sec	
Name	Causes	Spot size			Part-A	Part-B
Spot	B/W spot in glass or protection sheet, foreign mat. Pin hole	D ≤ 0.25mm			Ignore	Ignore
		0.25mm ＜ D ≤ 0.4mm			4	
		0.4mm ＜ D			0	
Scratch or line defect	Scratch on glass or Scratch on FPL or Particle is Protection sheet.	Length	Width		Part-A	Ignore
		L ≤2.0mm	W≤0.2 mm		Ignore	
		2.0 mm < L≤ 5.0mm	0.2 mm<W≤ 0.3mm		2	
		5.0 mm < L	0.3mm < W		0	
Air bubble	Air bubble	D1, D2 ≤ 0.2 mm			Ignore	Ignore
		0.2 mm < D1,D2 ≤ 0.35mm			4	
		0.35mm < D1, D2			0	
Side Fragment						

	X≤5mm，Y≤1mm & display is ok, Ignore

Remarks: Spot define: That only can be seen under WS or DS defects.

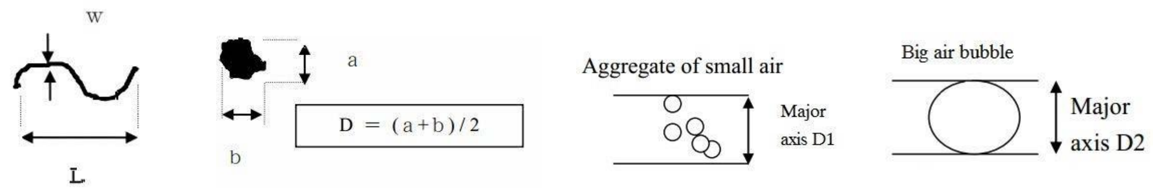
□ Any defect which is visible under gray pattern or transition process but invisible under black and white is disregarded.

□ Here is definition of the “Spot” and “Scratch or line defect”.

Spot: W > 1/4L   Scratch or line defect: W ≤1/4L

□ Definition for L/W and D (major axis)

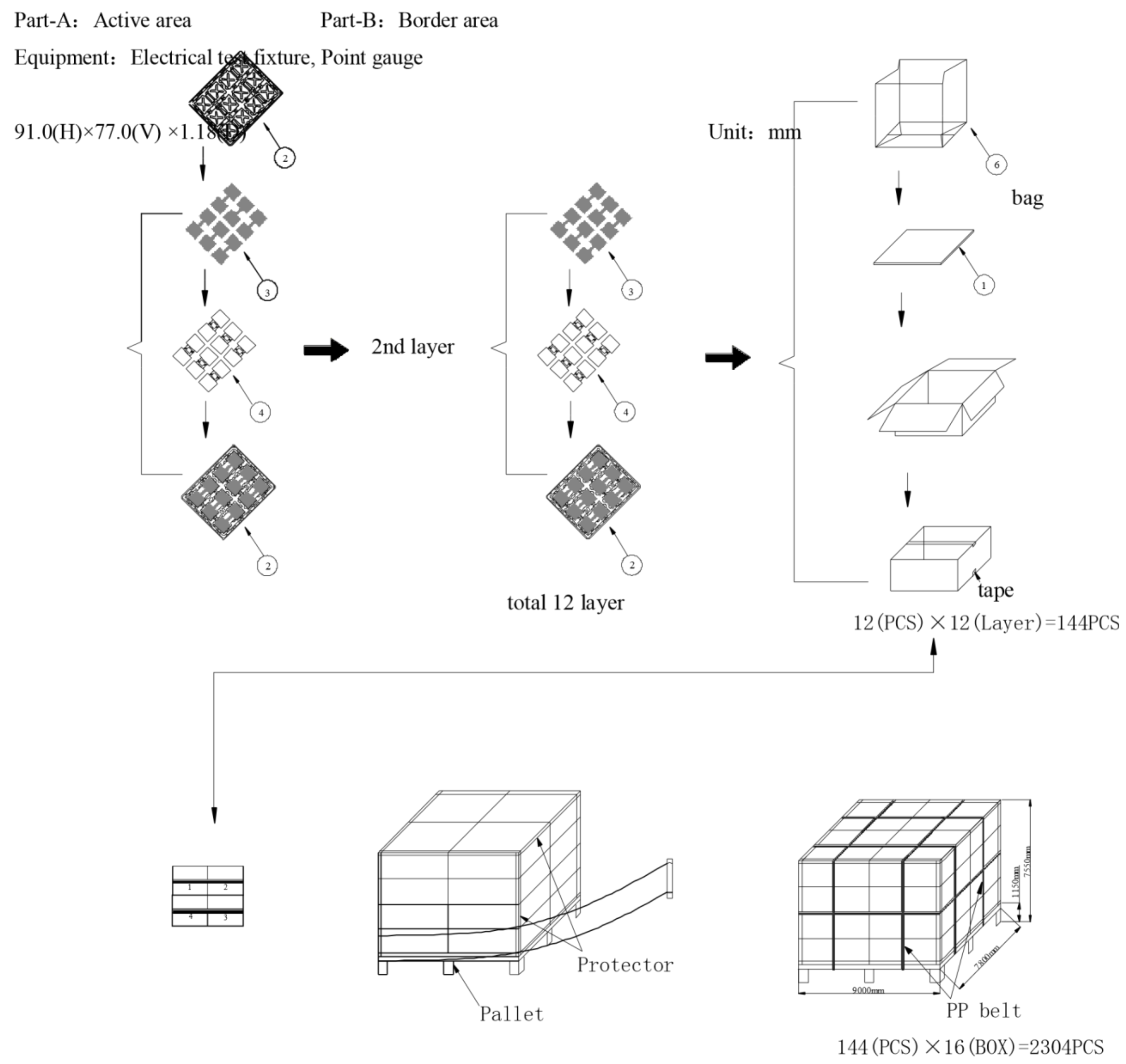
□ FPC bonding area pad doesn’t allowed visual inspection.



Note: AQL = 0.4

13. Packing

Shipment Inseption Standard



Outline dimensionempty tray：

vacuum

1st layer

Note: AQL = 0.4

Appendix G: Prototype Code

```
#include <Wire.h>
#include <PWFusion_TCA9548A.h>
#include "Adafruit_TCS34725.h"
#include <math.h>

//pressure sensor pins
#define Q1 39
#define Q2 34
#define Q3 35
#define Q4 32


// global variables
  //color sensors
    TCA9548A i2cMux;
    Adafruit_TCS34725 tcs;//8-bit var
    uint16_t red, green, blue, clear;//16-bit var

  //general
    uint16_t p1, p2, p3, p4, Q1val, Q2val, Q3val, Q4val, Q1zero, Q2zero, Q3zero, Q4zero,
bagLocation;
    int Q1diff, Q3diff, Q2diff, Q4diff;
    int percentRed, Clear, percentBlue;
    uint8_t error=0;
    int minVal=3000;

void setup()
{
  // Initialize I2C and Serial
  Serial.begin(9600);// begin comms. with serial monitor
  Serial.println("*****************************************");

  Wire.begin();
  i2cMux.begin(0x70);
  i2cMux.setChannel(CHAN_NONE);

  // Initialize color sensors
  i2cMux.begin(0x71);
  i2cMux.setChannel(CHAN1);//QUAD1
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
  Serial.println("\n");
  if (tcs.begin()) {
    Serial.println("Found sensor Q1");
  } else {
    Serial.println("No TCS34725 found color sensor found for Q1");
    error=1;
  }
  i2cMux.setChannel(CHAN2);//QUAD2
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
  if (tcs.begin()) {
    Serial.println("Found sensor Q2");
  } else {
    Serial.println("No TCS34725 found color sensor found for Q2");
    error=1;
  }
  //change from MUX001 to MUX000
  i2cMux.setChannel(CHAN_NONE);
  i2cMux.begin(0x70);

  i2cMux.setChannel(CHAN3);//QUAD3
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
  if (tcs.begin()) {
    Serial.println("Found sensor Q3");
  } else {
    Serial.println("No TCS34725 color sensor found for Q3");
    error=1;
  }
  i2cMux.setChannel(CHAN4);//QUAD4
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
  if (tcs.begin()) {
    Serial.println("Found sensor Q4");
  } else {
    Serial.println("No TCS34725 color sensor found for Q4");
    error=1;
  }
  i2cMux.setChannel(CHAN_NONE);
  //Wire.endTransmission();
  Wire.end();
  //Error loo
  while(error)
  {
    Serial.println("error");
    delay(10000);
  }


  //Calibrate pressure sensors
    analogReadResolution(12);
    //pressure sensor read
    Serial.println("Clear the board for calibration.");
    delay(5000);
    //QUAD 1
    Q1val=analogRead(Q1);
    //QUAD 2
    Q2val=analogRead(Q2);
    //QUAD 3
    Q3val=analogRead(Q3);
    //QUAD 4
    Q4val=analogRead(Q4);
  //ensure all feet have contact
  while(1)//calibration loop
  {
    if(Q1val>minVal)
    {
      Serial.print("Add shim to foot in Quad 1.");
      while(1)
      {
        Q1val=analogRead(Q1);
        if(Q1val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p1=1;}
    if(Q2val>minVal)
    {
      Serial.print("Add shim to foot in Quad 2.");
      while(1)
      {
        Q2val=analogRead(Q2);
        if(Q2val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p2=1;}
    if(Q3val>minVal)
    {
      Serial.print("Add shim to foot in Quad 3.");
      while(1)
      {
        Q3val=analogRead(Q3);
        if(Q3val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p3=1;}
    if(Q4val>minVal)
    {
      Serial.print("Add shim to foot in Quad 4.");
      while(1)
      {
        Q4val=analogRead(Q4);
        if(Q4val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p4=1;}

    if((p1+p2+p3+p4)==4){break;}//leave calibration loop

  }//end calibration loop
  //calibrate zeros
  Q1zero=Q1val;
  Q2zero=Q2val;
  Q3zero=Q3val;
  Q4zero=Q4val;
  Serial.println("Pressure sensors are calibrated.");

}//end setup

void loop() {
  //bag location logic
    //pressure sensor read
    //QUAD 1
    Q1val=analogRead(Q1);
    //Serial.print("Quadrant 1: ");Serial.print(Q1val);
    Q1diff=Q1zero-Q1val;
    Serial.print("Quadrant 1: ");Serial.print(Q1diff);
    //QUAD 2
    Q2val=analogRead(Q2);
    //Serial.print("Quadrant 2: ");Serial.print(Q2val);
    Q2diff=Q2zero-Q2val;
    Serial.print("Quadrant 2: ");Serial.print(Q2diff);
    //QUAD 3
    Q3val=analogRead(Q3);
    //Serial.print("Quadrant 3: ");Serial.print(Q3val);
    Q3diff=Q3zero-Q3val;
    Serial.print("Quadrant 3: ");Serial.print(Q3diff);
    //QUAD 4
    Q4val=analogRead(Q4);
    //Serial.print("Quadrant 4: ");Serial.print(Q4val);
    Q4diff=Q4zero-Q4val;
    Serial.print("Quadrant 4: ");Serial.print(Q4diff);
    //find bag location
    if(Q1diff<Q2diff && Q1diff<Q3diff && Q1diff<Q4diff)
    {
      bagLocation=3;
    }//end if
    else if(Q2diff<Q1diff && Q2diff<Q3diff && Q2diff<Q4diff)
    {
      bagLocation=4;
    }//end else if
    else if(Q3diff<Q1diff && Q3diff<Q2diff && Q3diff<Q4diff)
    {
      bagLocation=1;
    }//end else if
    else if(Q4diff<Q1diff && Q4diff<Q2diff && Q4diff<Q3diff)
    {
      bagLocation=2;
    }//end else if


    if (bagLocation==1)
    {
      Wire.begin();
      i2cMux.begin(0x71);
      i2cMux.setChannel(CHAN1);
    }//end if location 1
    if (bagLocation==2)
    {
      Wire.begin();
      i2cMux.begin(0x71);
      i2cMux.setChannel(CHAN2);
    }//end if location 2
    if (bagLocation==3)
    {
      Wire.begin();
      i2cMux.begin(0x70);
      i2cMux.setChannel(CHAN3);
    }//end if location 3
    if (bagLocation==4)
    {
      Wire.begin();
      i2cMux.begin(0x70);
      i2cMux.setChannel(CHAN4);
    }//end if location 4

      tcs.getRawData(&red, &green, &blue, &clear);
      Clear=clear/100;
      percentRed=red/Clear;
      percentBlue=blue/Clear;
      //Serial.print("Red= "); Serial.print(percentRed,DEC); Serial.print("     ");
      //Serial.print("Blue= "); Serial.print(percentBlue,DEC); Serial.println("     ");


      if(percentRed>70)
      {
        Serial.println("The board has no bags.");
      }//end if
      else if(percentRed<=percentBlue+5)
      {
        Serial.print("A blue bag is in quadrant ");Serial.print(bagLocation); Serial.println(".");
      }//end else if
      else if(percentRed>(percentBlue+5))
      {
        Serial.print("A red bag is in quadrant ");Serial.print(bagLocation); Serial.println(".");
      }//end else if

       delay(3000);
}//end loop

```

Appendix H: Primary Communication Code

```
#include <esp_now.h>
#include <WiFi.h>
#include <esp_wifi.h> // only for esp_wifi_set_channel()

// Global copy of slave
esp_now_peer_info_t slave;
#define CHANNEL 1
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0

struct DataPacket{

int redscore;
int bluescore;

};

DataPacket dataToSend;

const int buttonPin = 2;

void setup() {
  Serial.begin(115200);
  //Set device in STA mode to begin with
  WiFi.mode(WIFI_STA);
  esp_wifi_set_channel(CHANNEL, WIFI_SECOND_CHAN_NONE);
  Serial.println("ESPNow/Basic/Master Example");
  // This is the mac address of the Master in Station Mode
  Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());
  Serial.print("STA CHANNEL "); Serial.println(WiFi.channel());
  // Init ESPNow with a fallback logic
  InitESPNow();
  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  //Scores
  dataToSend.redscore = 2;
  dataToSend.bluescore = 2;

  pinMode(buttonPin, INPUT_PULLUP);

}

// Init ESP Now with fallback
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    // InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// Scan for slaves in AP mode
void ScanForSlave() {
  int16_t scanResults = WiFi.scanNetworks(false, false, false, 300, CHANNEL); // Scan only on one channel
  // reset on each scan
  bool slaveFound = 0;
  memset(&slave, 0, sizeof(slave));

  Serial.println("");
  if (scanResults == 0) {
    Serial.println("No WiFi devices in AP Mode found");
  } else {
    Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
    for (int i = 0; i < scanResults; ++i) {
      // Print SSID and RSSI for each device found
      String SSID = WiFi.SSID(i);
      int32_t RSSI = WiFi.RSSI(i);
      String BSSIDstr = WiFi.BSSIDstr(i);

      if (PRINTSCANRESULTS) {
        Serial.print(i + 1);
        Serial.print(": ");
        Serial.print(SSID);
        Serial.print(" (");
        Serial.print(RSSI);
        Serial.print(")");
        Serial.println("");
      }
      delay(10);
      // Check if the current device starts with `Slave`
      if (SSID.indexOf("Slave") == 0) {
        // SSID of interest
        Serial.println("Found a Slave.");
        Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" ["); Serial.print(BSSIDstr); Serial.print("]"); Serial.print(" ("); Serial.print(RSSI); Serial.print(")"); Serial.println("");
        // Get BSSID => Mac Address of the Slave
        int mac[6];
        if ( 6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x",  &mac[0], &mac[1], &mac[2], &mac[3], &mac[4], &mac[5] ) ) {
          for (int ii = 0; ii < 6; ++ii ) {
            slave.peer_addr[ii] = (uint8_t) mac[ii];
          }
        }

        slave.channel = CHANNEL; // pick a channel
        slave.encrypt = 0; // no encryption

        slaveFound = 1;
        // we are planning to have only one slave in this example;
        // Hence, break after we find one, to be a bit efficient
        break;
      }
    }
  }

  if (slaveFound) {
    Serial.println("Slave Found, processing..");
  } else {
    Serial.println("Slave Not Found, trying again.");
  }

  // clean up ram
  WiFi.scanDelete();
}

// Check if the slave is already paired with the master.
// If not, pair the slave with master
bool manageSlave() {
  if (slave.channel == CHANNEL) {
    if (DELETEBEFOREPAIR) {
      deletePeer();
    }

    Serial.print("Slave Status: ");
    // check if the peer exists
    bool exists = esp_now_is_peer_exist(slave.peer_addr);
    if ( exists) {
      // Slave already paired.
      Serial.println("Already Paired");
      return true;
    } else {
      // Slave not paired, attempt pair
      esp_err_t addStatus = esp_now_add_peer(&slave);
      if (addStatus == ESP_OK) {
        // Pair success
        Serial.println("Pair success");
        return true;
      } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
        // How did we get so far!!
        Serial.println("ESPNOW Not Init");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
        Serial.println("Invalid Argument");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
        Serial.println("Peer list full");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
        Serial.println("Out of memory");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {
        Serial.println("Peer Exists");
        return true;
      } else {
        Serial.println("Not sure what happened");
        return false;
      }
    }
  } else {
    // No slave found to process
    Serial.println("No Slave found to process");
    return false;
  }
}

void deletePeer() {
  esp_err_t delStatus = esp_now_del_peer(slave.peer_addr);
  Serial.print("Slave Delete Status: ");
  if (delStatus == ESP_OK) {
    // Delete success
    Serial.println("Success");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW Not Init");
  } else if (delStatus == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found.");
  } else {
    Serial.println("Not sure what happened");
  }
}

// send data
void sendData() {

  const uint8_t *peer_addr = slave.peer_addr;
  Serial.print("Sending"); //Serial.println(dataToSend);
  esp_err_t result = esp_now_send(peer_addr, (uint8_t *)&dataToSend, sizeof(dataToSend));
  Serial.print("Send Status: ");
  if (result == ESP_OK) {
    Serial.println("Success");
  } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW not Init.");
  } else if (result == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
    Serial.println("Internal Error");
  } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
    Serial.println("ESP_ERR_ESPNOW_NO_MEM");
  } else if (result == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found.");
  } else {
    Serial.println("Not sure what happened");
  }

  if (digitalRead(buttonPin) == LOW)
  {
    dataToSend.redscore++;
  }
  if (digitalRead(buttonPin) == LOW)
  {
    dataToSend.bluescore++;
  }
  //dataToSend.redscore++;
  //dataToSend.bluescore++;
}

// callback when data is sent from Master to Slave
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  Serial.print("Last Packet Send Status: "); Serial.println(status == ESP_NOW_SEND_SUCCESS ?
"Delivery Success" : "Delivery Fail");
}

void loop() {
  // In the loop we scan for slave
  ScanForSlave();
  // If Slave is found, it would be populate in `slave` variable
  // We will check if `slave` is defined and then we proceed further
  if (slave.channel == CHANNEL) { // check if slave channel is defined
    // `slave` is defined
    // Add slave as peer if it has not been added already
    bool isPaired = manageSlave();
    if (isPaired) {
      // pair success or already paired
      // Send data to device
      sendData();
    } else {
      // slave pair failed
      Serial.println("Slave pair failed!");
    }
  }
  else {
    // No slave found to process
  }

  // wait for 1 second to run the logic again
  delay(5000);
}

```

Appendix I: Secondary Communication Code

```
#include <esp_now.h>
#include <WiFi.h>

#define CHANNEL 1

int receivedValue1;
int receivedValue2;

// Init ESP Now with fallback
void InitESPNow() {
  WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    // InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// config AP SSID
void configDeviceAP() {
  const char *SSID = "Slave_1";
  bool result = WiFi.softAP(SSID, "Slave_1_Password", CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
    Serial.print("AP CHANNEL "); Serial.println(WiFi.channel());
  }
}

void setup() {
  Serial.begin(115200);
  Serial.println("ESPNow/Basic/Slave Example");
  //Set device in AP mode to begin with
  WiFi.mode(WIFI_AP);
  // configure device AP mode
  configDeviceAP();
  // This is the mac address of the Slave in AP Mode
  Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
  // Init ESPNow with a fallback logic
  InitESPNow();
  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info.
  esp_now_register_recv_cb(OnDataRecv);
}

struct DataPacket {
  int value1;
  int value2;
};

// callback when data is recv from Master
void OnDataRecv(const uint8_t *mac_addr, const uint8_t *data, int data_len) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Recv from: "); Serial.println(macStr);
  //Serial.print("Last Packet Recv Data: "); Serial.println(*data);
  Serial.println("");
  DataPacket* packet = (DataPacket*)data;
  int receivedValue1 = packet->value1;
  int receivedValue2 = packet->value2;



  Serial.print("Red Score is ");
  Serial.println(receivedValue1);
  Serial.print("Blue Score is ");
  Serial.println(receivedValue2);

    if(receivedValue1 >= 21)
  {
    Serial.print("Red team Wins");
  }

  if(receivedValue2 >= 21)
  {
    Serial.print("Blue team Wins");
  }

}



void loop() {
  // Chill

}

```

Appendix J: Gameplay Code

```
// Test values were added to progress the gameplay

#include <Wire.h>
```

```
#include <PWFusion_TCA9548A.h>
#include "Adafruit_TCS34725.h"
#include <math.h>
#include <esp_now.h>
#include <WiFi.h>
#include <esp_wifi.h> // only for esp_wifi_set_channel()
#include <GxEPD.h>
#include <GxGDEW042T2/GxGDEW042T2.h>    // 4.2" b/w
#include <Fonts/FreeMonoBold24pt7b.h>
#include <GxIO/GxIO_SPI/GxIO_SPI.h>
#include <GxIO/GxIO.h>

// Global copy of slave
esp_now_peer_info_t slave;
#define CHANNEL 1
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0

//pressure sensor pins
#define Q1 26
#define Q2 27
#define Q3 14
#define Q4 12

#define PWR1 39
#define PWR2 34
#define PWR3 35
#define PWR4 32
#define on HIGH
#define off LOW

int Q1zero; int Q2zero; int Q3zero; int Q4zero;
int Q1val; int Q2val; int Q3val; int Q4val;
int Q5val; int Q6val; int Q7val; int Q8val;
int Q9val; int Q10val; int Q11val; int Q12val;
int Q13val; int Q14val; int Q15val; int Q16val;
int Q17val; int Q18val; int Q19val; int Q20val;
int Q21val; int Q22val; int Q23val; int Q24val;
int Q25val; int Q26val; int Q27val; int Q28val;
int Q29val; int Q30val; int Q31val; int Q32val;
int Q1diff; int Q2diff; int Q3diff; int Q4diff; int bag1[4];
int Q5diff; int Q6diff; int Q7diff; int Q8diff; int bag2[4];
int Q9diff; int Q10diff; int Q11diff; int Q12diff; int bag3[4];
int Q13diff; int Q14diff; int Q15diff; int Q16diff; int bag4[4];
int Q17diff; int Q18diff; int Q19diff; int Q20diff; int bag5[4];
int Q21diff; int Q22diff; int Q23diff; int Q24diff; int bag6[4];
int Q25diff; int Q26diff; int Q27diff; int Q28diff; int bag7[4];
int Q29diff; int Q30diff; int Q31diff; int Q32diff; int bag8[4];
int minVal = 3000; int Turn = 1; int bagLocation;
//u_int8_t round = 0;
int bagsensedonboard = 0; int bagsensedinhole = 0;
int p1, p2, p3, p4;
int Team1Score = 0;
int Team2Score = 0;
int Team1ScoreCurrent = 0;
int Team2ScoreCurrent = 0;
const int buttonPin1 = 13;
const int buttonPin2 = 33;

struct DataPacket
{
int Team1Score;
int Team2Score;
};

```

```
DataPacket dataToSend;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);// begin comms. with serial monitor
  Serial.println("*****************************************");
      analogReadResolution(12);
    //pressure sensor read
    Serial.println("Clear the board for calibration.");
    delay(5000);
    //QUAD 1
    Q1val=analogRead(Q1);
    //QUAD 2
    Q2val=analogRead(Q2);
    //QUAD 3
    Q3val=analogRead(Q3);
    //QUAD 4
    Q4val=analogRead(Q4);
  //ensure all feet have contact
 while(1)//calibration loop
  {
    if(Q1val>minVal)
    {
      Serial.print("Add shim to foot in Quad 1.");
      while(1)
      {
        Q1val=analogRead(Q1);
        if(Q1val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p1=1;}
    if(Q2val>minVal)
    {
      Serial.print("Add shim to foot in Quad 2.");
      while(1)
      {
        Q2val=analogRead(Q2);
        if(Q2val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p2=1;}
    if(Q3val>minVal)
    {
      Serial.print("Add shim to foot in Quad 3.");
      while(1)
      {
        Q3val=analogRead(Q3);
        if(Q3val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p3=1;}
    if(Q4val>minVal)
    {
      Serial.print("Add shim to foot in Quad 4.");
      while(1)
      {
        Q4val=analogRead(Q4);
        if(Q4val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p4=1;}

    if((p1+p2+p3+p4)==4){break;}//leave calibration loop

  }//end calibration loop
  Serial.println("out of loop");
  //calibrate zeros
  Q1zero=Q1val;
  Q2zero=Q2val;
  Q3zero=Q3val;
  Q4zero=Q4val;

  Serial.println(Q1zero);
  Serial.println(Q2zero);
  Serial.println(Q3zero);
  Serial.println(Q4zero);

  // Set button pins as inputs
  pinMode(buttonPin1, INPUT);
  pinMode(buttonPin2, INPUT);
}

void pointcalculation()
{

  if (Team1ScoreCurrent > Team2ScoreCurrent)
  {
    Team1Score = Team1Score + Team1ScoreCurrent - Team2ScoreCurrent;
    Team1ScoreCurrent = Team2ScoreCurrent = 0;
  }
  else if (Team1ScoreCurrent < Team2ScoreCurrent)
  {
    Team2Score = Team2Score + Team2ScoreCurrent - Team1ScoreCurrent;
    Team1ScoreCurrent = Team2ScoreCurrent = 0;
  }
  else
  {
    Team1Score = Team1Score;
    Team2Score = Team2Score;
    Team1ScoreCurrent = Team2ScoreCurrent = 0;
  }
}

void endgame()
{
  if (Team1Score >= 21 && Team2Score < 21)
  {
    Serial.println("Team 1 wins");
    Team1Score = Team2Score = 0; //resets score
  }
  else if (Team2Score >= 21 && Team1Score < 21)
  {
    Serial.println("Team 2 wins");
    Team1Score = Team2Score = 0; //resets score
  }
}

void loop() {
  // put your main code here, to run repeatedly:
if(Turn==1)
{
//bag 1
  if (digitalRead(buttonPin1) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q1val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q2val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q3val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q4val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q1diff = Q1val-Q1zero;
    Q2diff = Q2val-Q2zero;
    Q3diff = Q3val-Q3zero;
    Q4diff = Q4val-Q4zero;

    bag1[1] = Q1diff; //bag 1 location
    bag1[2] = Q2diff;
    bag1[3] = Q3diff;
    bag1[4] = Q4diff;

    if(Q1diff<Q2diff && Q1diff<Q3diff && Q1diff<Q4diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
      // Turn on quad 3 color sensor mux
    }//end if
    else if(Q2diff<Q1diff && Q2diff<Q3diff && Q2diff<Q4diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
      // Turn on quad 4 color sensor mux
    }//end else if
    else if(Q3diff<Q1diff && Q3diff<Q2diff && Q3diff<Q4diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
      // Turn on quad 1 color sensor mux
    }//end else if
    else if(Q4diff<Q1diff && Q4diff<Q2diff && Q4diff<Q3diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
      // Turn on quad 2 color sensor mux
    }//end else if
    else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 3;
    }

    Team1ScoreCurrent = Team1ScoreCurrent+1;
    Turn = Turn+1;
    Serial.println(Q1diff);
    Serial.println(Q2diff);
    Serial.println(Q3diff);
    Serial.println(Q4diff);
    Serial.println(bagLocation);
    Serial.println(Team1ScoreCurrent);
    Serial.println("It is team 2 turn");
    delay(500);
  }

}
//bag 2
if(Turn==2)
{
//bag 2
  if(digitalRead(buttonPin2) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q5val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q6val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q7val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4,on);
    delay(250);
    Q8val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q5diff = Q5val-Q1val;
    Q6diff = Q6val-Q2val;
    Q7diff = Q7val-Q3val;
    Q8diff = Q8val-Q4val;

    bag2[1] = Q5diff; //bag 2 location
    bag2[2] = Q6diff;
    bag2[3] = Q7diff;
    bag2[4] = Q8diff;

    if(Q5diff<Q6diff && Q5diff<Q7diff && Q5diff<Q8diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q6diff<Q5diff && Q6diff<Q7diff && Q6diff<Q8diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q7diff<Q5diff && Q7diff<Q6diff && Q7diff<Q8diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q8diff<Q5diff && Q8diff<Q6diff && Q8diff<Q7diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
    else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 3;
    }

    Turn = Turn+1;
    Serial.println(Q5diff);
    Serial.println(Q6diff);
    Serial.println(Q7diff);
    Serial.println(Q8diff);
    Serial.println(bagLocation);
    Serial.println(Team2ScoreCurrent);
    Serial.println("It is team 1 turn");
    delay(500);
  }

}
//bag 3
if(Turn==3)
{
//bag 3
  if(digitalRead(buttonPin1) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q9val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q10val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q11val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q12val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q9diff = Q9val-Q5val;
    Q10diff = Q10val-Q6val;
    Q11diff = Q11val-Q7val;
    Q12diff = Q12val-Q8val;

    bag3[1] = Q9diff; //bag 3 location
    bag3[2] = Q10diff;
    bag3[3] = Q11diff;
    bag3[4] = Q12diff;

    if(Q9diff<Q10diff && Q9diff<Q11diff && Q9diff<Q12diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q10diff<Q9diff && Q10diff<Q11diff && Q10diff<Q12diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q11diff<Q9diff && Q11diff<Q10diff && Q11diff<Q12diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q12diff<Q9diff && Q12diff<Q10diff && Q12diff<Q11diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 3;
    }
    Team1ScoreCurrent = Team1ScoreCurrent+1;

    Turn = Turn+1;
    Serial.println(bagLocation);
    Serial.println(Team1ScoreCurrent);
    Serial.println("It is team 2 turn");
    delay(500);
  }

}
//bag 4
if(Turn==4)
{
//bag 4
  if(digitalRead(buttonPin2) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q13val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q14val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q15val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q16val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q13diff = Q13val-Q9val;
    Q14diff = Q14val-Q10val;
    Q15diff = Q15val-Q11val;
    Q16diff = Q16val-Q12val;

    bag4[1] = Q13diff; //bag 4 location
    bag4[2] = Q14diff;
    bag4[3] = Q15diff;
    bag4[4] = Q16diff;

    if(Q13diff<Q14diff && Q13diff<Q15diff && Q13diff<Q16diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q15diff<Q13diff && Q15diff<Q14diff && Q15diff<Q16diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q15diff<Q13diff && Q15diff<Q14diff && Q15diff<Q16diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q16diff<Q13diff && Q16diff<Q14diff && Q16diff<Q15diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 3;
    }

    Turn = Turn+1;
    Serial.println(bagLocation);
    Serial.println(Team2ScoreCurrent);
    Serial.println("It is team 1 turn");
    delay(500);
  }

}

if(Turn==5)
{
//bag 5
  if(digitalRead(buttonPin1) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q17val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q18val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q19val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q20val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q17diff = Q17val-Q13val; //-Q13Val?
    Q18diff = Q18val-Q14val;
    Q19diff = Q19val-Q15val;
    Q20diff = Q20val-Q16val;

    bag5[1] = Q17diff; //bag 5 location
    bag5[2] = Q18diff;
    bag5[3] = Q19diff;
    bag5[4] = Q20diff;

    if(Q17diff<Q18diff && Q17diff<Q19diff && Q17diff<Q20diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q18diff<Q17diff && Q18diff<Q19diff && Q18diff<Q20diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q19diff<Q17diff && Q19diff<Q18diff && Q19diff<Q20diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q20diff<Q17diff && Q20diff<Q18diff && Q20diff<Q19diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 3;
    }

    Team1ScoreCurrent = Team1ScoreCurrent+1;
    Turn = Turn+1;
    Serial.println(bagLocation);
    Serial.println(Team1ScoreCurrent);
    Serial.println("It is team 2 turn");
    delay(500);
  }

}

if(Turn==6)
{
//bag 6
  if(digitalRead(buttonPin2) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q21val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q22val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q23val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q24val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q21diff = Q21val-Q17val;
    Q22diff = Q22val-Q18val;
    Q23diff = Q23val-Q19val;
    Q24diff = Q24val-Q20val;

    bag6[1] = Q21diff; //bag 6 location
    bag6[2] = Q22diff;
    bag6[3] = Q23diff;
    bag6[4] = Q24diff;

    if(Q21diff<Q22diff && Q21diff<Q23diff && Q21diff<Q24diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q22diff<Q21diff && Q22diff<Q23diff && Q22diff<Q24diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q23diff<Q21diff && Q23diff<Q2diff && Q23diff<Q24diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q24diff<Q21diff && Q24diff<Q22diff && Q24diff<Q23diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 3;
    }

    Turn = Turn+1;
    Serial.println(bagLocation);
    Serial.println(Team2ScoreCurrent);
    Serial.println("It is team 1 turn");
    delay(500);
  }

 }

if(Turn==7)
```

```
{
//bag 7
  if(digitalRead(buttonPin1) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q25val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q26val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q27val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q28val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q25diff = Q25val-Q21val;
    Q26diff = Q26val-Q22val;
    Q27diff = Q27val-Q23val;
    Q28diff = Q28val-Q24val;

    bag7[1] = Q25diff; //bag 7 location
    bag7[2] = Q26diff;
    bag7[3] = Q27diff;
    bag7[4] = Q28diff;

    if(Q25diff<Q26diff && Q25diff<Q27diff && Q25diff<Q28diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q26diff<Q25diff && Q26diff<Q27diff && Q26diff<Q28diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q27diff<Q25diff && Q27diff<Q26diff && Q27diff<Q28diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q28diff<Q25diff && Q28diff<Q26diff && Q28diff<Q27diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 3;
    }

    Team1ScoreCurrent = Team1ScoreCurrent+1;
    Turn = Turn+1;
    Serial.println(bagLocation);
    Serial.println(Team1ScoreCurrent);
    Serial.println("It is team 2 turn");
    delay(500);
  }

}

if(Turn==8)
{
//bag 8
  if(digitalRead(buttonPin2) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q29val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q30val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q31val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q32val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q29diff = Q29val-Q25val;
    Q30diff = Q30val-Q26val;
    Q31diff = Q31val-Q27val;
    Q32diff = Q32val-Q28val;

    bag8[1] = Q29diff; //bag 4 location
    bag8[2] = Q30diff;
    bag8[3] = Q31diff;
    bag8[4] = Q32diff;

    if(Q29diff<Q30diff && Q29diff<Q31diff && Q29diff<Q32diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
    }//end if
    else if(Q30diff<Q29diff && Q30diff<Q31diff && Q30diff<Q32diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
    }//end else if
    else if(Q31diff<Q29diff && Q31diff<Q30diff && Q31diff<Q32diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
    }//end else if
    else if(Q32diff<Q29diff && Q32diff<Q30diff && Q32diff<Q31diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
    }//end else if
        else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team2ScoreCurrent = Team2ScoreCurrent + 3;
    }

   // round = round + 1;
    Serial.println(bagLocation);
    Serial.println(Team2ScoreCurrent);
    Serial.println("End of round");
    delay(500);

  }
    pointcalculation();

if(Turn==8)
{
//bag 7
  if(digitalRead(buttonPin1) == HIGH)
  {
  Serial.print("Red teams score is "); Serial.println(Team1Score);
  Serial.print("Blue teams score is "); Serial.println(Team2Score);

  scoreboard(Team1Score, Team2Score, &FreeMonoBold24pt7b);
  scoreboardNumbers(Team1Score, Team2Score, &FreeMonoBold24pt7b);

  endgame();

  Turn = 1;
  Serial.print(Turn);
  }

}
}
}

```

Appendix K: Button Test

```
/* ***************************************************************
Cornhole Board Button Test
Summary:
This code turns on and off the built-in LED when the button is pushed.
Date: 11/25/23
Name: Tim Desser
*************************************************************** */
const int buttonPin = 33;  // the number of the pushbutton pin
const int ledPin = 2;    // the number of the built-in LED pin

// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}

```

Appendix L: Single Color Sensor Test

```
/* ************************************************************
Color Sensor Single Test
Summary:
Uses MUX to find one color sensors
Date: 11/21/23
Name: Tim Desser
************************************************************* */

#include <Wire.h>
#include <PWFusion_TCA9548A.h>
#include "Adafruit_TCS34725.h"
#include <math.h>




// global variables
  //color sensors
    TCA9548A i2cMux;
    Adafruit_TCS34725 tcs;//8-bit var
    uint16_t red, green, blue, clear;//16-bit var

void setup()
{
  // Initialize I2C and Serial
  Serial.begin(9600);// begin comms. with serial monitor
  delay(5000);
  Serial.println("****************************************");

  Wire.begin();

  // contact/calibrate color sensors
  // hole sensors
  i2cMux.begin(0X72);
  Serial.println("For Q3 MUX:");
  i2cMux.setChannel(CHAN0);
  delay(500);
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
  Serial.println();
  if (tcs.begin()) {
    Serial.println("sensor 0 found");
  } else {
    Serial.println("No TCS34725 found for sensor 0");
  }
  tcs.getRawData(&red, &green, &blue, &clear);
  Serial.println("setup");
  Serial.print("red=");
  Serial.println(red);
  Serial.print("green=");
  Serial.println(green);
  Serial.print("blue=");
  Serial.println(blue);
  Serial.print("white=");
  Serial.println(clear);
  delay(5000);

  i2cMux.setChannel(CHAN_NONE);
  //Wire.endTransmission();
  Wire.end();
}//end void setup
void loop(){

   Wire.begin();
  // contact/calibrate color sensors
  // hole sensors
   i2cMux.begin(0X72);

  // contact/calibrate color sensors
  // hole sensors

  Serial.println("For Q3 MUX:");
  i2cMux.setChannel(CHAN1);
  tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);

  if (tcs.begin()) {
    Serial.println("sensor 1 found");
  } else {
    Serial.println("No TCS34725 found for sensor 1");
  }
  tcs.getRawData(&red, &green, &blue, &clear);
  Serial.print("red=");
  Serial.println(red);
  Serial.print("green=");
  Serial.println(green);
  Serial.print("blue=");
  Serial.println(blue);
  Serial.print("white=");
  Serial.println(clear);
  Serial.println();
  delay(5000);


}

```

Appendix M: Multiple Color Sensors Test

```
/* *************************************************************
Color Sensor Find Test
Summary:
Uses MUX to find all color sensors
Date: 11/21/23
Name: Tim Desser
************************************************************* */
```

```

#include <Wire.h>
#include <PWFusion_TCA9548A.h>
#include "Adafruit_TCS34725.h"
#include <math.h>

//pressure sensor pins
#define Qh 0x70
#define Q1 0x71
#define Q2 0x72
#define Q3 0x73
#define Q4 0x74


// global variables
  //color sensors
    TCA9548A i2cMux;
    Adafruit_TCS34725 tcs;//8-bit var
    uint16_t red, green, blue, clear;//16-bit var

void setup()
{
  // Initialize I2C and Serial
  Serial.begin(9600);// begin comms. with serial monitor
  Serial.println("****************************************");

  Wire.begin();
  delay(5000);
  //hole MUX off
  i2cMux.begin(0x70);
  i2cMux.setChannel(CHAN_NONE);
  //Q1 MUX off
  i2cMux.begin(0x71);
  i2cMux.setChannel(CHAN_NONE);
  //Q2 MUX off
  i2cMux.begin(0x72);
  i2cMux.setChannel(CHAN_NONE);
  //Q3 MUX off
  i2cMux.begin(0x73);
  i2cMux.setChannel(CHAN_NONE);
  //Q4 MUX off
  i2cMux.begin(0x74);
  i2cMux.setChannel(CHAN_NONE);

  // contact/calibrate color sensors
  // hole sensors
   i2cMux.begin(0x70);
   Serial.println("For Hole MUX:");
   i2cMux.setChannel(CHAN0);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   Serial.println("\n");
   if (tcs.begin()) {
     Serial.println("sensor 0 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 0");
   }
   i2cMux.setChannel(CHAN1);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 1 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 1");
   }
   i2cMux.setChannel(CHAN7);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 7 found");
   } else {
     Serial.println("No TCS34725 found color sensor found for sensor 7");
   }
  // //end hole MUX

  // //change to Q1 Mux
   i2cMux.setChannel(CHAN_NONE);
   i2cMux.begin(Q1);
   Serial.println("For Q1 MUX:");

   i2cMux.setChannel(CHAN0);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 0 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 0");
   }
   i2cMux.setChannel(CHAN1);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 1 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 1");
   }
    i2cMux.setChannel(CHAN2);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 2 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 2");
   }
    i2cMux.setChannel(CHAN3);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 3 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 3");
   }
    i2cMux.setChannel(CHAN4);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 4 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 4");
   }
    i2cMux.setChannel(CHAN5);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 5 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 5");
   }
    i2cMux.setChannel(CHAN6);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 6 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 6");
   }
    i2cMux.setChannel(CHAN7);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 7 found");
```

```
     } else {
       Serial.println("No TCS34725 color sensor found for sensor 7");
     }
     i2cMux.setChannel(CHAN_NONE);
     //end Q1 MUX

   //change to Q2 Mux
   i2cMux.setChannel(CHAN_NONE);
   i2cMux.begin(Q2);
   Serial.println("For Q2 MUX:");

   i2cMux.setChannel(CHAN0);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 0 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 0");
   }
   i2cMux.setChannel(CHAN1);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 1 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 1");
   }
    i2cMux.setChannel(CHAN2);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 2 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 2");
   }
    i2cMux.setChannel(CHAN3);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 3 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 3");
   }
    i2cMux.setChannel(CHAN4);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 4 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 4");
   }
    i2cMux.setChannel(CHAN5);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 5 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 5");
   }
    i2cMux.setChannel(CHAN6);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 6 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 6");
   }
    i2cMux.setChannel(CHAN7);
   tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
   if (tcs.begin()) {
     Serial.println("sensor 7 found");
   } else {
     Serial.println("No TCS34725 color sensor found for sensor 7");
   }
   i2cMux.setChannel(CHAN_NONE);
   //end Q2 MUX

    //change to Q3 Mux
    i2cMux.setChannel(CHAN_NONE);
    i2cMux.begin(0x73);
    Serial.println("For Q3 MUX:");

    i2cMux.setChannel(CHAN0);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 0 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 0");
    }
    i2cMux.setChannel(CHAN1);
      tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 1 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 1");
    }
     i2cMux.setChannel(CHAN2);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 2 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 2");
    }
     i2cMux.setChannel(CHAN3);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 3 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 3");
    }
     i2cMux.setChannel(CHAN4);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 4 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 4");
    }
     i2cMux.setChannel(CHAN5);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 5 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 5");
    }
     i2cMux.setChannel(CHAN6);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 6 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 6");
    }
     i2cMux.setChannel(CHAN7);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 7 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 7");
    }
    i2cMux.setChannel(CHAN_NONE);
  end Q3 MUX

  //change to Q4 Mux
    i2cMux.setChannel(CHAN_NONE);
    i2cMux.begin(Q4);
    Serial.println("For Q4 MUX:");

    i2cMux.setChannel(CHAN0);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 0 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 0");
    }
    i2cMux.setChannel(CHAN1);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 1 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 1");
    }
     i2cMux.setChannel(CHAN2);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 2 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 2");
    }
     i2cMux.setChannel(CHAN3);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 3 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 3");
    }
     i2cMux.setChannel(CHAN4);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 4 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 4");
    }
     i2cMux.setChannel(CHAN5);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 5 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 5");
    }
     i2cMux.setChannel(CHAN6);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 6 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 6");
    }
     i2cMux.setChannel(CHAN7);
    tcs=Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_614MS, TCS34725_GAIN_60X);
    if (tcs.begin()) {
      Serial.println("sensor 7 found");
    } else {
      Serial.println("No TCS34725 color sensor found for sensor 7");
    }
```

```
   i2cMux.setChannel(CHAN_NONE);
   //end Q4 MUX

   i2cMux.setChannel(CHAN_NONE);
   //Wire.endTransmission();
   Wire.end();
}//end void setup
void loop(){}

```

Appendix N: Pressure Sensors Test

```
// Pressure sensor testing

#include <Wire.h>
#include <PWFusion_TCA9548A.h>
#include "Adafruit_TCS34725.h"
#include <math.h>
#include <esp_now.h>
#include <WiFi.h>
#include <esp_wifi.h> // only for esp_wifi_set_channel()
#include <GxEPD.h>
#include <GxGDEW042T2/GxGDEW042T2.h>    // 4.2" b/w
#include <Fonts/FreeMonoBold24pt7b.h>
#include <GxIO/GxIO_SPI/GxIO_SPI.h>
#include <GxIO/GxIO.h>

// Global copy of slave
esp_now_peer_info_t slave;
#define CHANNEL 1
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0

//pressure sensor pins
#define Q1 26
#define Q2 27
#define Q3 14
#define Q4 12

#define PWR1 39
#define PWR2 34
#define PWR3 35
#define PWR4 32
#define on HIGH
#define off LOW
int Q1zero; int Q2zero; int Q3zero; int Q4zero;
int Q1val; int Q2val; int Q3val; int Q4val;
int Q1diff; int Q2diff; int Q3diff; int Q4diff; int bag1[4];
int p1, p2, p3, p4;
int Team1Score = 0;
int Team2Score = 0;
int Team1ScoreCurrent = 0;
int Team2ScoreCurrent = 0;
const int buttonPin1 = 13;
const int buttonPin2 = 33;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);// begin comms. with serial monitor
  Serial.println("*****************************************");
      analogReadResolution(12);
    //pressure sensor read
    Serial.println("Clear the board for calibration.");
    delay(5000);
    //QUAD 1
    Q1val=analogRead(Q1);
    //QUAD 2
    Q2val=analogRead(Q2);
    //QUAD 3
    Q3val=analogRead(Q3);
    //QUAD 4
    Q4val=analogRead(Q4);
  //ensure all feet have contact
 while(1)//calibration loop
  {
    if(Q1val>minVal)
    {
      Serial.print("Add shim to foot in Quad 1.");
      while(1)
      {
        Q1val=analogRead(Q1);
        if(Q1val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p1=1;}
    if(Q2val>minVal)
    {
      Serial.print("Add shim to foot in Quad 2.");
      while(1)
      {
        Q2val=analogRead(Q2);
        if(Q2val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p2=1;}
    if(Q3val>minVal)
    {
      Serial.print("Add shim to foot in Quad 3.");
      while(1)
      {
        Q3val=analogRead(Q3);
        if(Q3val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p3=1;}
    if(Q4val>minVal)
    {
      Serial.print("Add shim to foot in Quad 4.");
      while(1)
      {
        Q4val=analogRead(Q4);
        if(Q4val<minVal){break;}
      }//wait for user adjustment
    }//end if
    else{p4=1;}

    if((p1+p2+p3+p4)==4){break;}//leave calibration loop

  }//end calibration loop
  Serial.println("out of loop");
  //calibrate zeros
  Q1zero=Q1val;
  Q2zero=Q2val;
  Q3zero=Q3val;
  Q4zero=Q4val;

  Serial.println(Q1zero);
  Serial.println(Q2zero);
  Serial.println(Q3zero);
  Serial.println(Q4zero);

  // Set button pins as inputs
  pinMode(buttonPin1, INPUT);
  pinMode(buttonPin2, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
if(Turn==1)
{
//bag 1
  if (digitalRead(buttonPin1) == HIGH)
  {
    digitalWrite(PWR1, on);
    delay(250);
    Q1val = analogRead(Q1);
    digitalWrite(PWR1, off);
    digitalWrite(PWR2, on);
    delay(250);
    Q2val = analogRead(Q2);
    digitalWrite(PWR2, off);
    digitalWrite(PWR3, on);
    delay(250);
    Q3val = analogRead(Q3);
    digitalWrite(PWR3, off);
    digitalWrite(PWR4, on);
    delay(250);
    Q4val = analogRead(Q4);
    digitalWrite(PWR4, off);

    Q1diff = Q1val-Q1zero;
    Q2diff = Q2val-Q2zero;
    Q3diff = Q3val-Q3zero;
    Q4diff = Q4val-Q4zero;

    bag1[1] = Q1diff; //bag 1 location
    bag1[2] = Q2diff;
    bag1[3] = Q3diff;
    bag1[4] = Q4diff;

    if(Q1diff<Q2diff && Q1diff<Q3diff && Q1diff<Q4diff)
    {
      bagLocation=3;
      bagsensedonboard = 1;
      // Turn on quad 3 color sensor mux
    }//end if
    else if(Q2diff<Q1diff && Q2diff<Q3diff && Q2diff<Q4diff)
    {
      bagLocation=4;
      bagsensedonboard = 1;
      // Turn on quad 4 color sensor mux
    }//end else if
    else if(Q3diff<Q1diff && Q3diff<Q2diff && Q3diff<Q4diff)
    {
      bagLocation=1;
      bagsensedonboard = 1;
      // Turn on quad 1 color sensor mux
    }//end else if
    else if(Q4diff<Q1diff && Q4diff<Q2diff && Q4diff<Q3diff)
    {
      bagLocation=2;
      bagsensedonboard = 1;
      // Turn on quad 2 color sensor mux
    }//end else if
    else
    {
      bagsensedonboard = 0;
    }

    if (bagsensedonboard == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 1;
    }
    else if (bagsensedinhole == 1)
    {
    Team1ScoreCurrent = Team1ScoreCurrent + 3;
    }

    Serial.println(Q1diff);
    Serial.println(Q2diff);
    Serial.println(Q3diff);
    Serial.println(Q4diff);
}

```

Appendix O: LED On/Off Test

```
/* ****************************************************************
Cornhole Board LED on/off Test
Summary:
This code turns on and off the LED back lights.
Date: 11/25/23
Name: Tim Desser
************************************************************** */

#define on HIGH
#define off LOW
void setup() {

  pinMode(LED, OUTPUT);
}


void loop() {
  digitalWrite(LED, on);
  delay(5000);
  digitalWrite(LED, off);
  delay(1000);
}

```

Appendix P: LED PWM Test

```
/* ****************************************************************
Cornhole Board LED
Summary:
This code controls the LED lights on the cornhole board using a PWM signal.
Date: 11/25/23
```

```
Name: Tim Desser
****************************************************************** */
#include "ESP32_FastPWM.h"
#define LED 15
int PWM_resolution       = 12;
//creates pwm instance
ESP32_FAST_PWM* PWM_Instance;

float frequency = 1000.0f;
float dutyCycle = 0.0f;
int del = 20;
uint8_t channel = 0;

void setup()
{
 //assigns PWM frequency of 1.0 KHz and a duty cycle of 0%, channel 0, 12-bit resolution
  PWM_Instance = new ESP32_FAST_PWM(LED, frequency, dutyCycle, channel, PWM_resolution);
}

void loop()
{
  for(int i=0;i<100;i++)
  {
    PWM_Instance->setPWM(LED, frequency, i);
    delay(del);
  }
  for(int i=100;i>0;i--)
  {
    PWM_Instance->setPWM(LED, frequency, i);
    delay(del);
  }

  // PWM_Instance->setPWM(LED, frequency, 90);
  // delay(del);
  // PWM_Instance->setPWM(LED, frequency, 10);
  // delay(del);

}

```

Appendix Q: Solenoid Test

```
/* ******************************************************************
Cornhole Board Solenoid Test
Summary:
This code controls the solenoid for the hole door.
Date: 11/25/23
Name: Tim Desser
****************************************************************** */
#define sol 2
#define on HIGH
#define off LOW
void setup() {

  pinMode(sol, OUTPUT);
}


void loop() {
  digitalWrite(sol, on);
  delay(250);
  digitalWrite(sol, off);
  delay(10000);
}
```

```
```

Appendix R: Display Test

```
/* **************************************************************
Cornhole Board display Test
Summary:
This code displays two different fonts and creates dividing lines.
Date: 11/25/23
Name: Tim Desser
************************************************************** */
//** First, install the GxEPD libray and move custom fonts to the fonts folder.
#include <GxEPD.h>
#include <GxGDEW042T2/GxGDEW042T2.h>      // 4.2" b/w
#include <Fonts/Nokora_Bold_100.h>
#include <Fonts/FreeMonoBold24pt7b.h>
#include <GxIO/GxIO_SPI/GxIO_SPI.h>
#include <GxIO/GxIO.h>

GxIO_Class io(SPI, /*CS=5*/ SS, /*DC=*/ 17, /*RST=*/ 16); // arbitrary selection of 17, 16
GxEPD_Class display(io, /*RST=*/ 16, /*BUSY=*/ 4); // arbitrary selection of (16), 4
int RedScore=5;
int BlueScore=9;
void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println("setup");
  display.init(115200); // enable diagnostic output on Serial
  Serial.println("setup done");
}

void loop()
{
  scoreboard(RedScore, BlueScore, &FreeMonoBold24pt7b);
  scoreboardNumbers(RedScore, BlueScore, &Nokora_Bold_100);
  delay(30000);
  RedScore++;
  BlueScore++;
}

void scoreboard(int redScore, int blueScore , const GFXfont* f)
{
  display.fillScreen(GxEPD_WHITE);
  display.setTextColor(GxEPD_BLACK);
  display.setFont(f);
  display.setCursor(5, 60);
  display.print("CORNHOLE");
  display.setCursor(30, 130);
  display.print("RED");
  display.setCursor(240, 130);
  display.print("BLUE");

  // draw lines
  display.fillRect(0, 70, 400, 4, GxEPD_BLACK);
  display.fillRect(198, 70, 4, 230, GxEPD_BLACK);


}

void scoreboardNumbers(int redScore, int blueScore , const GFXfont* f)
{
  //display.fillScreen(GxEPD_WHITE);
  //display.setTextColor(GxEPD_BLACK);
  display.setFont(f);
  // display.setCursor(5, 60);
  // display.print("CORNHOLE");
  // display.setCursor(30, 130);
  // display.print("RED");
  // display.setCursor(210, 130);
  // display.print("BLUE");
  if(redScore<=9){
    display.setCursor(70, 250);}//end if
  else{
    display.setCursor(50, 250);}//end else
  display.print(redScore);
  if(blueScore<=9){
    display.setCursor(270, 250);}//end if
  else{
    display.setCursor(250, 250);}//end else
  display.print(blueScore);

  display.update();
  //delay(1000);

}

```