IOT Smart Plant Monitoring and Irrigation System Project

Ahmed Abdelgawad	201901249
Mahmoud Sheliel	201900697
Mario Youchia	201902086
Yahya Ashraf	202000776

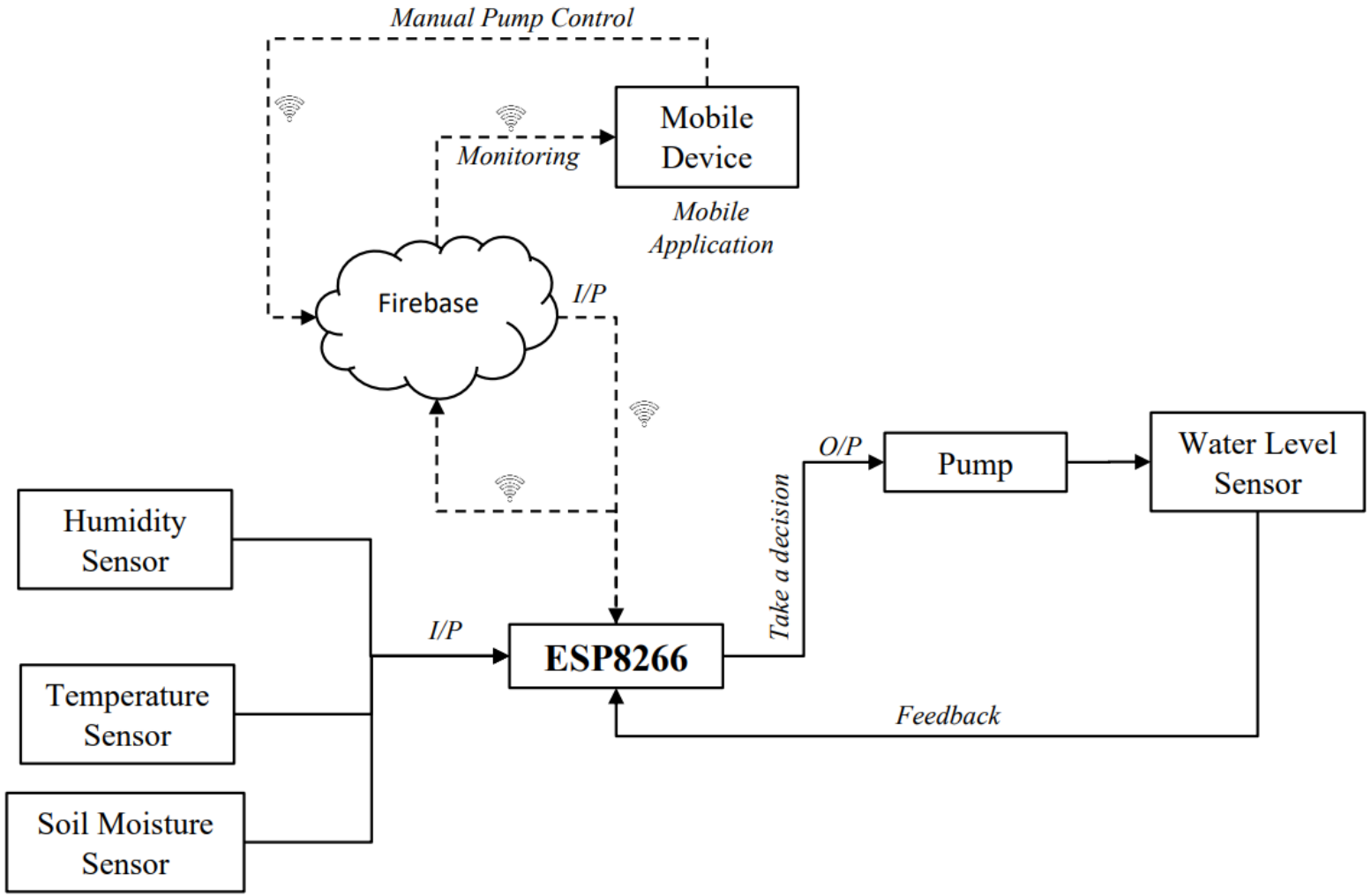
Introduction

An IoT smart plant monitoring and irrigation system is a technology that helps people monitor and water their plants using the internet. The system includes sensors that measure factors like soil moisture, temperature, and humidity, and then sends that information to the firebase that can be accessed online. Users can then check on their plants remotely, and the system can even water the plants automatically based on the data collected by the sensors.

Introduction

There are many different types of smart plant monitoring and irrigation systems available, ranging from DIY projects like this project that use basic components, to commercial products that are designed specifically for agriculture. These systems can be a great way to keep plants healthy, even if the user is not able to be physically present to care for them.

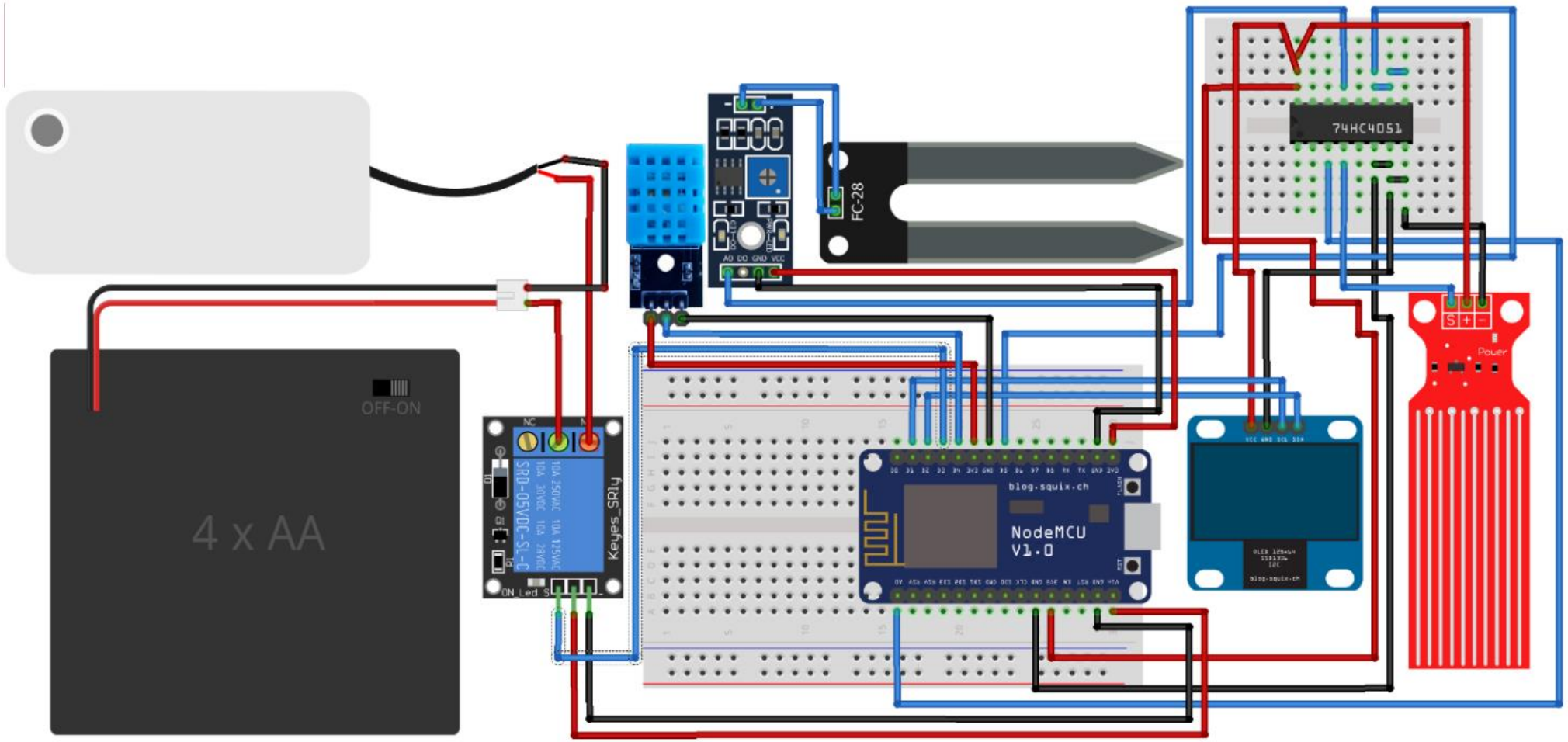
Functional Block Diagram



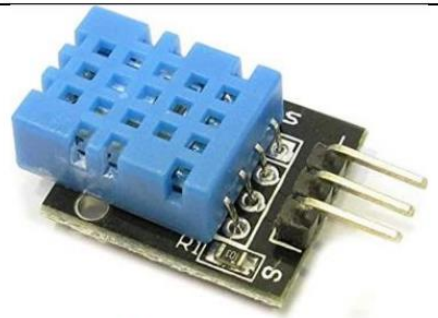
Functional Block Diagram

The system begins by measuring the temperature, humidity of the environment, and the soil moisture through the corresponding sensors. These readings are used by the ESP8266 NodeMCU board to take a decision whether to open the pump or not. The water level sensor checks continuously if the water level exceeds some value, if it exceeds this value, the pump will be turned off. Also, the ESP8266 sends the sensors readings to the firebase, so the mobile application can read these values and monitor the irrigation system.

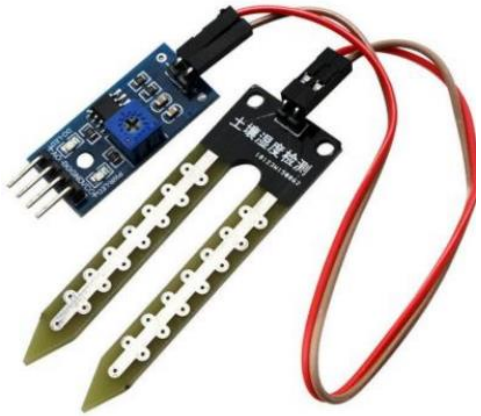
Schematic Diagram



Components

Component	Function
 ESP8266 NodeMCU board	It is the microcontroller used to control the system, sends, and receives data to and from the firebase enabling real-time monitoring, and manual control.
 DHT11 Sensor	It is used to measure the humidity and temperature of the environment.

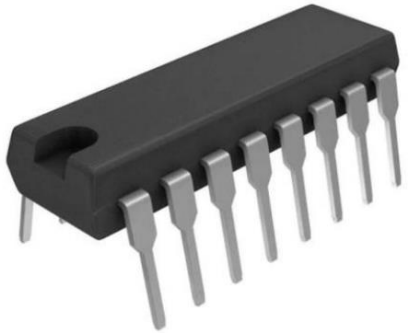
Components

Component	Function
 Soil Moisture Sensor	It is used to measure the moisture of the soil.
 Relay Module 5V – 1 Channel	It is used to turn ON or OFF the pump.

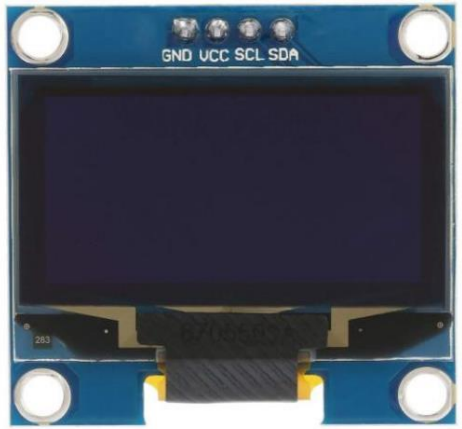
Components

Component	Function
 4x AA Battery	It is used to deliver power to the pump when the relay's control signal is high.
 DC pump – 6V	It is used to irrigate the plant.

Components

Component	Function
 CD74HC4051 – 8 Channels Multiplexer/Demultiplexer	It is used to enable the multi-use of the single analog pin available in the NodeMCU board.
 Water Level Sensor	It is used to measure the level of the water in the plant.

Components

Component	Function
 OLED 128x64 pixels	It is used to monitor the readings of sensors.

List of Components

Component	Price of One Unit	Quantity	Component Price
4x AA battery	20	1	20
Battery Holder	15	1	15
Pump + Hoses	85	1	85
Breadboard	25	1	25
ESP8266	200	1	200
Relay	50	1	50
DHT11	50	1	1
Soil Moisture Sensor	80	1	80
Tie Cables	1	10	10
M3x10 Bolts	1	20	20
M3 Nuts	1	20	20

List of Components

Component	Price of One Unit	Quantity	Component Price
Plant	25	1	25
Spacers	2.5	4	10
Laser Cut	100	1	100
Jumpers	1	15	15
Multiplexer	45	1	45
Water Level Sensor	25	1	25

Expected Budget: 795 EGP

Using Multiplexer or Arduino UNO with ESP?

Using a multiplexer with the ESP8266 that has only one analog input pin can provide several benefits, as follow:

- Firstly, it can allow for the expansion of the number of analog inputs available to the ESP8266, which is useful when the project requires multiple analog sensors to be monitored.
- Secondly, it can help to reduce the number of pins required for interfacing with the ESP8266, which is important in space-constrained designs.
- Thirdly, it can simplify the wiring and reduce the complexity of the code required to read from multiple sensors.

Using Multiplexer or Arduino UNO with ESP?

Additionally, by using the Arduino UNO with six analog pins and connecting it to the ESP8266, it allows for the flexibility of using the more powerful Arduino board to handle the analog input while still being able to communicate with the ESP8266 wirelessly. However, this requires an additional board and can be more complex to set up.

In contrast, using a multiplexer with the ESP8266 NodeMCU can provide a more compact and cost-effective solution, as well as lower power consumption. It can also be more suitable for certain applications where space is limited and a smaller number of analog inputs are required

Using Multiplexer or Arduino UNO with ESP?

Using a multiplexer with the ESP8266 to directly read from sensors can eliminate the need for data transfer between the two devices, potentially reducing overall latency and increasing the speed of data acquisition.

Using a multiplexer with the ESP8266 to directly read from sensors can eliminate the need for data transfer between the two devices, potentially reducing overall latency and increasing the speed of data acquisition.

According to the table shown, the total delay in reading output from multiplexer in worst case after changing the select pins would be 90 + 340 + 340 ns = 770 ns.

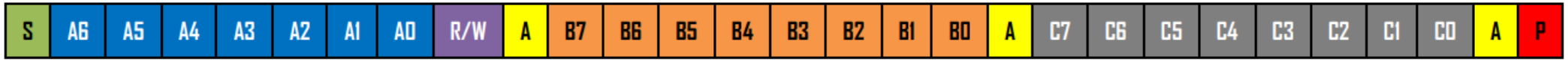
6.7 Switching Characteristics

over recommended operating free-air temperature range (unless otherwise noted) (see Figure 9)

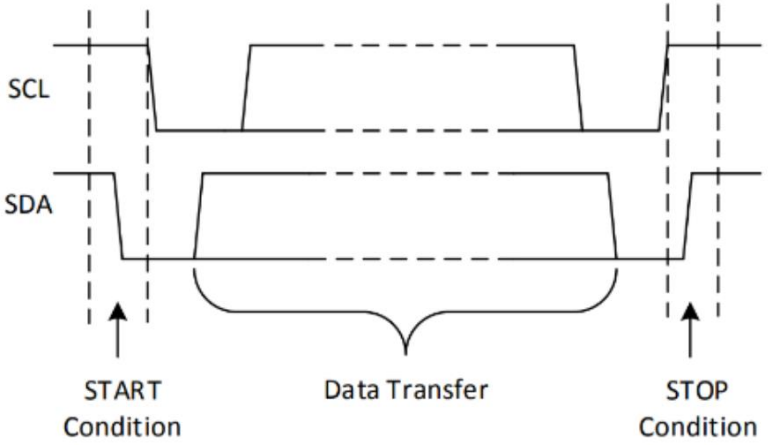
PARAMETER	FROM (INPUT)	TO (OUTPUT)	LOAD CAPACITANCE	V_{EE}	V_{CC}	T_A = 25°C			T_A = –55°C TO 125°C			UNIT
						MIN	TYP	MAX	MIN	TYP	MAX	
t_{pd}	IN	OUT	C_L = 15 pF		5 V			4				ns
			C_L = 50 pF	0 V	2 V			60			90	
					4.5 V			12			18	
					6 V			10			15	
				–4.5 V	4.5 V			8			12	
t_{en}	ADDRESS SEL or $\overline{E}$	OUT	C_L = 15 pF		5 V			19				ns
			C_L = 50 pF	0 V	2 V			225			340	
					4.5 V			45			68	
					6 V			38			57	
				–4.5 V	4.5 V			32			48	
t_{dis}	ADDRESS SEL or $\overline{E}$	OUT	C_L = 15 pF		5 V			19				ns
			C_L = 50 pF	0 V	2 V			225			340	
					4.5 V			45			68	
					6 V			38			57	
				–4.5 V	4.5 V			32			48	
C_I	Control		C_L = 50 pF					10			10	pF

Using Multiplexer or Arduino UNO with ESP?: I2C Protocol

In the case of sending data from Arduino UNO to ESP8266 using the I2C protocol, we would need at least 29 clock cycles.



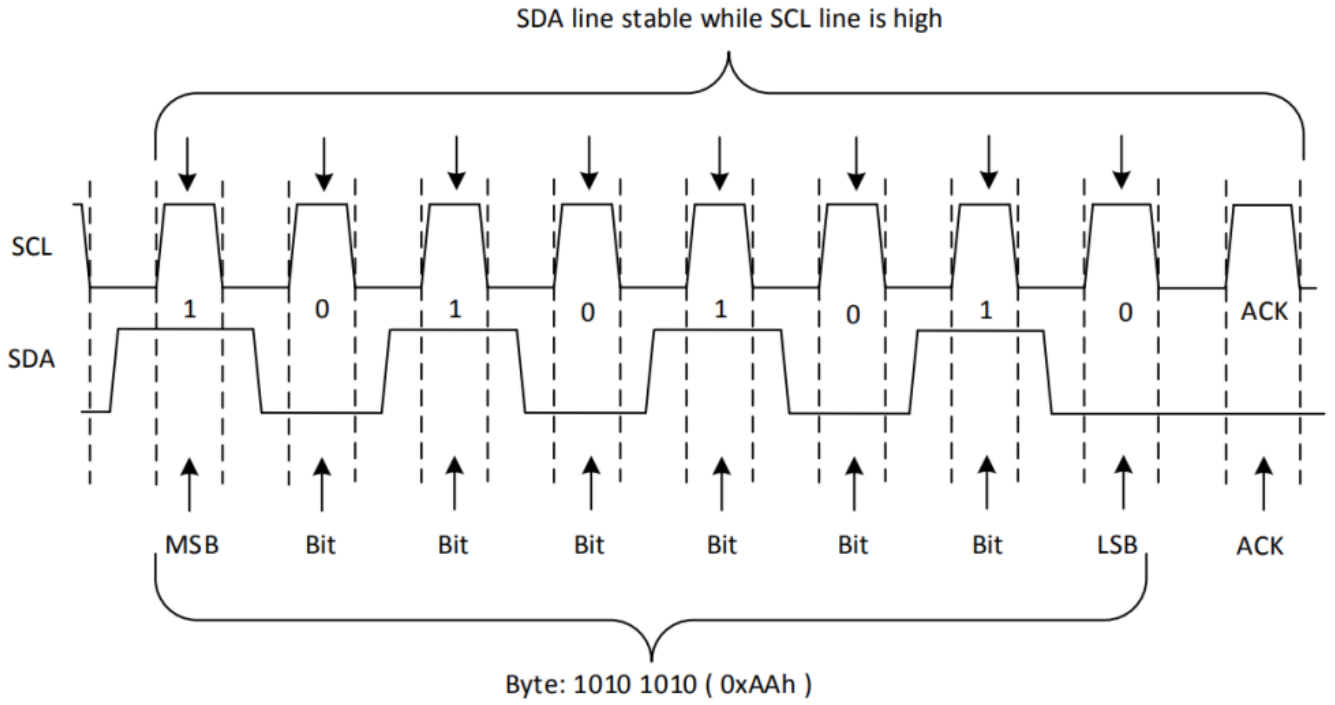
S: Start Condition - **P**: Stop Condition. The start condition is a high-to-low transition on the SDA line while the SCL is high, while the stop condition is a low-to-high transition on the SDA line while the SCL is high.



Using Multiplexer or Arduino UNO with ESP?: I2C Protocol

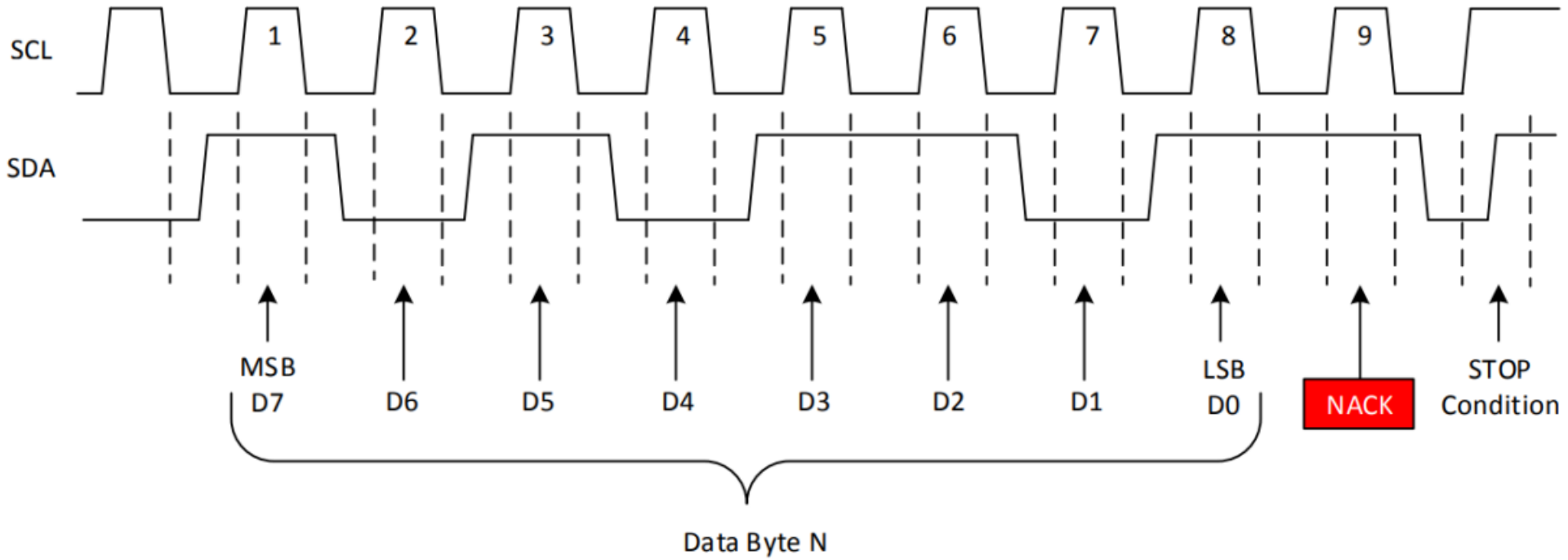
-A6-A0 bits are the address bits of the slave. - R/W is the read or write bit. - A is the acknowledgement bit. The address of the slave begins with the MSB, and ends with the LSB, each bit is transferred during one clock cycle.

The R/W bit indicates whether the master will read from the slave or write to it. If it is high, it means that the master will read from the slave, and if it is low, it means that the master will write to the slave. The A bit is an acknowledgement sent by the slave to indicate that it has received the preceding byte successfully.



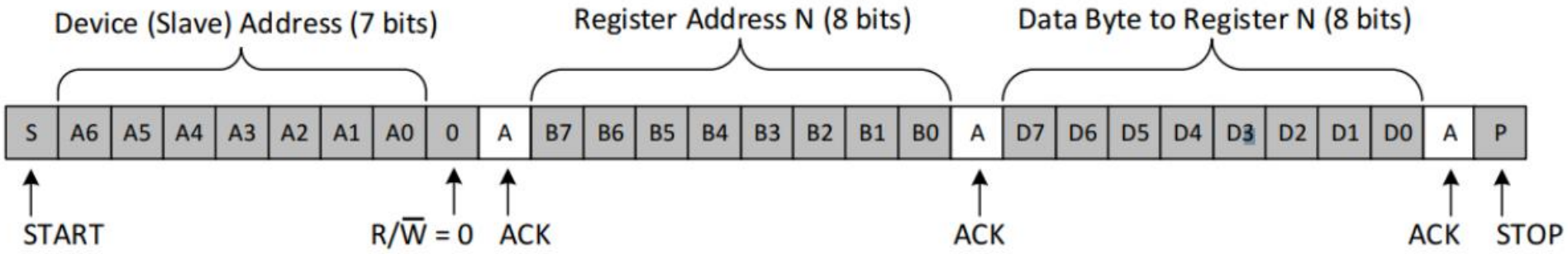
Using Multiplexer or Arduino UNO with ESP?: I2C Protocol

In case of ACK: the SDA line during the corresponding clock cycle will be pulled down, while in NACK it will be pulled high, then the sending process will be stopped.



Using Multiplexer or Arduino UNO with ESP?: I2C Protocol

- B7-B0: are the bits of the register address from which the data will be read, or to which the data will be written.
- C7-C0: are the bits of the read/written data.



Using Multiplexer or Arduino UNO with ESP?: I2C Protocol

Using the default I2C clock speed, which is 100kHz. Then, one clock cycle is equivalent to 10 microseconds.

Therefore, the 29 clock cycles take 290 microseconds.

$$290\mu s \gg 770ns$$

This is 376.6 times in the best case greater than the worst-case latency of using a multiplexer with an ESP8266 NodeMCU.

Arduino UNO vs ESP8266

Processing Power of Arduino UNO vs Processing Power of ESP8266 NodeMCU:

The processing power of the ESP8266 is higher than that of the Arduino Uno. The ESP8266 is equipped with a 32-bit RISC processor, running at 80 MHz, while the Arduino Uno is equipped with an 8-bit AVR processor, running at 16 MHz.

Additionally, the ESP8266 has more memory than the Arduino Uno, with up to 96 KB of RAM and 4 MB of flash memory available. This means that the ESP8266 can handle more complex tasks and programs than the Arduino Uno.

In summary, the primary difference between a 32-bit RISC (Reduced Instruction Set Computing) processor and an 8-bit AVR processor is their word size and instruction set architecture, which can affect their speed and performance in different types of applications

Arduino UNO vs ESP8266

- **ATMega328P** Processor
  - **Memory**
    - AVR CPU at up to 16 MHz
    - 32KB Flash
    - 2KB SRAM
    - 1KB EEPROM

Arduino UNO

3.1. CPU, Memory, and Flash

3.1.1. CPU

The ESP8266EX integrates a Tensilica L106 32-bit RISC processor, which achieves extra-low power consumption and reaches a maximum clock speed of 160 MHz. The Real-Time Operating System (RTOS) and Wi-Fi stack allow 80% of the processing power to be available for user application programming and development. The CPU includes the interfaces as below:

Ram:

According to our current version of SDK, SRAM space available to users is assigned as below.

- RAM size < 50 kB, that is, when ESP8266EX is working under the Station mode and connects to the router, the maximum programmable space accessible in Heap + Data section is around 50 kB.

ESP8266

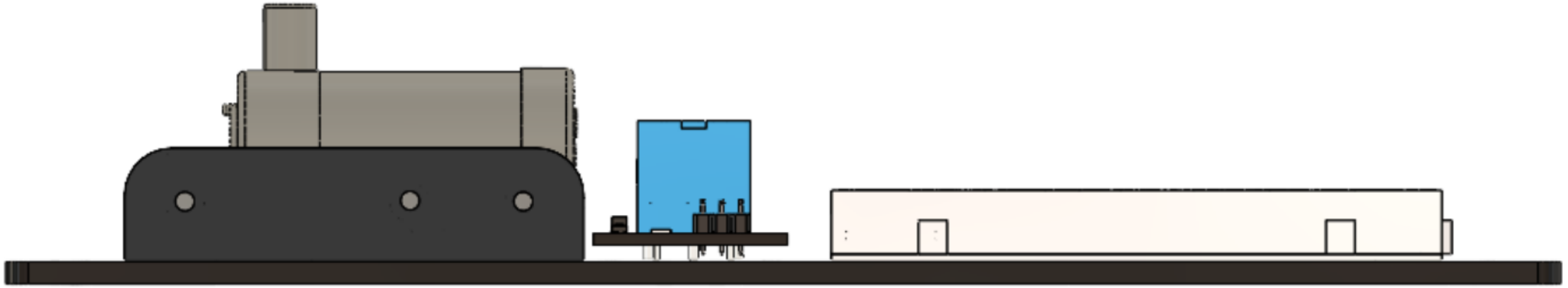
Conclusion

∵ The processing power of ESP8266 is higher than that of Arduino UNO, and it takes much more time to send data from Arduino UNO to ESP8266 using the I2C communication protocol.

∴ It is much wiser to **use a multiplexer** with ESP8266, which is much cheaper, and smaller than Arduino UNO board

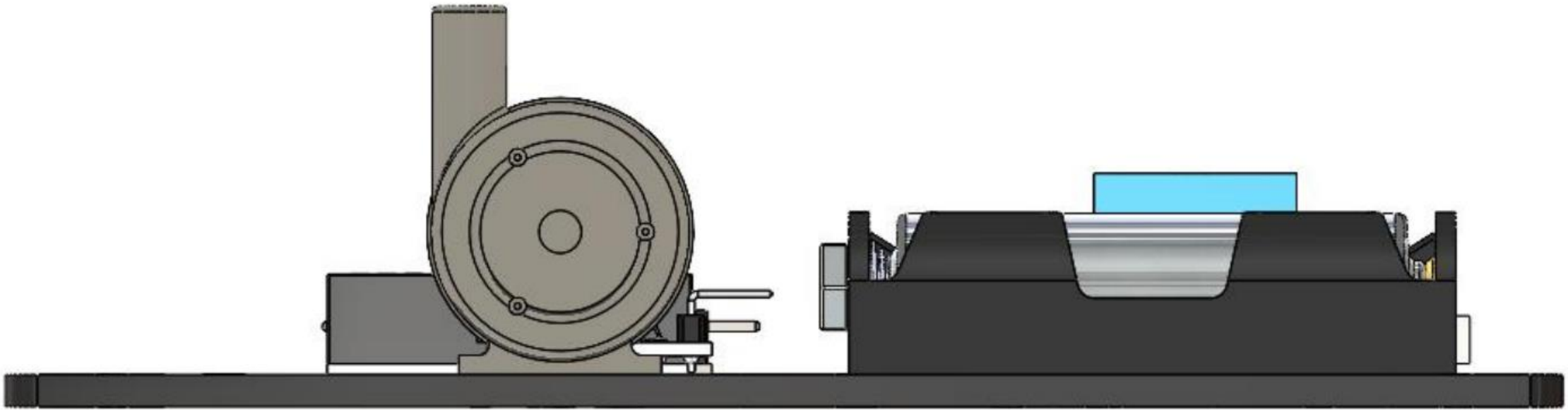
CAD Design

Elevation View:



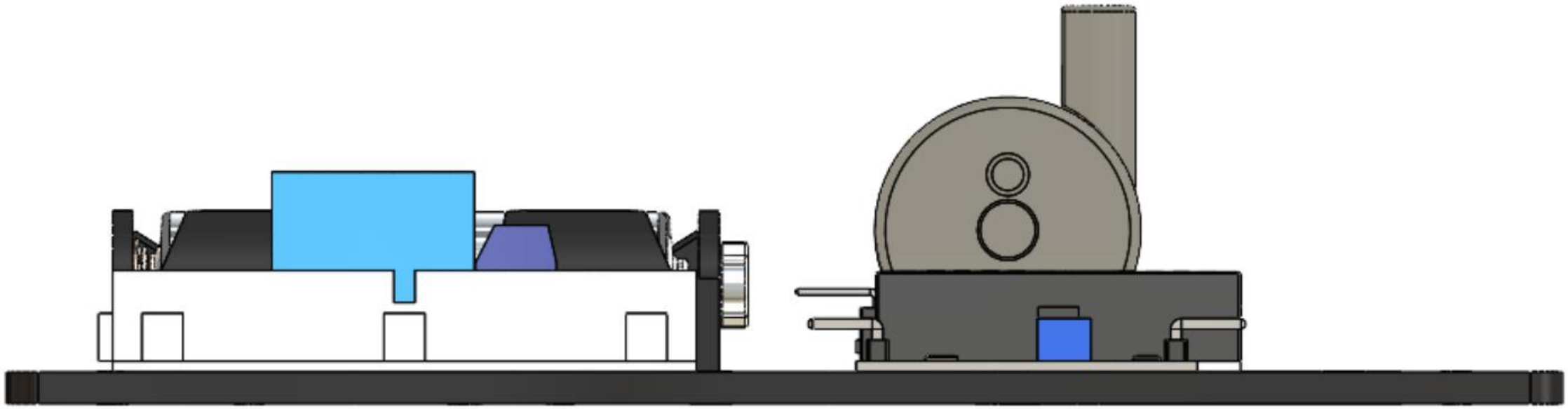
CAD Design

Left Side View:



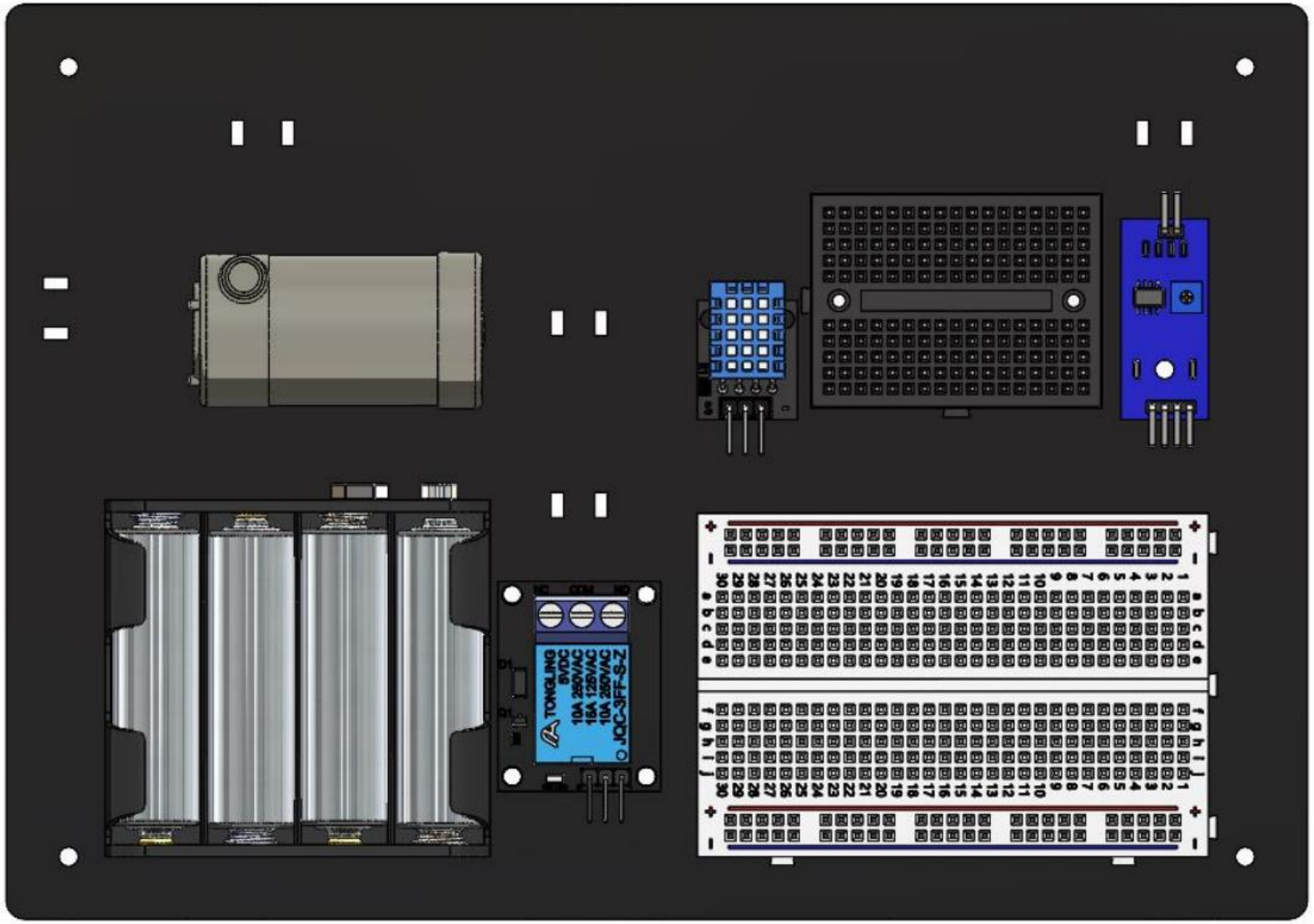
CAD Design

Right Side View:



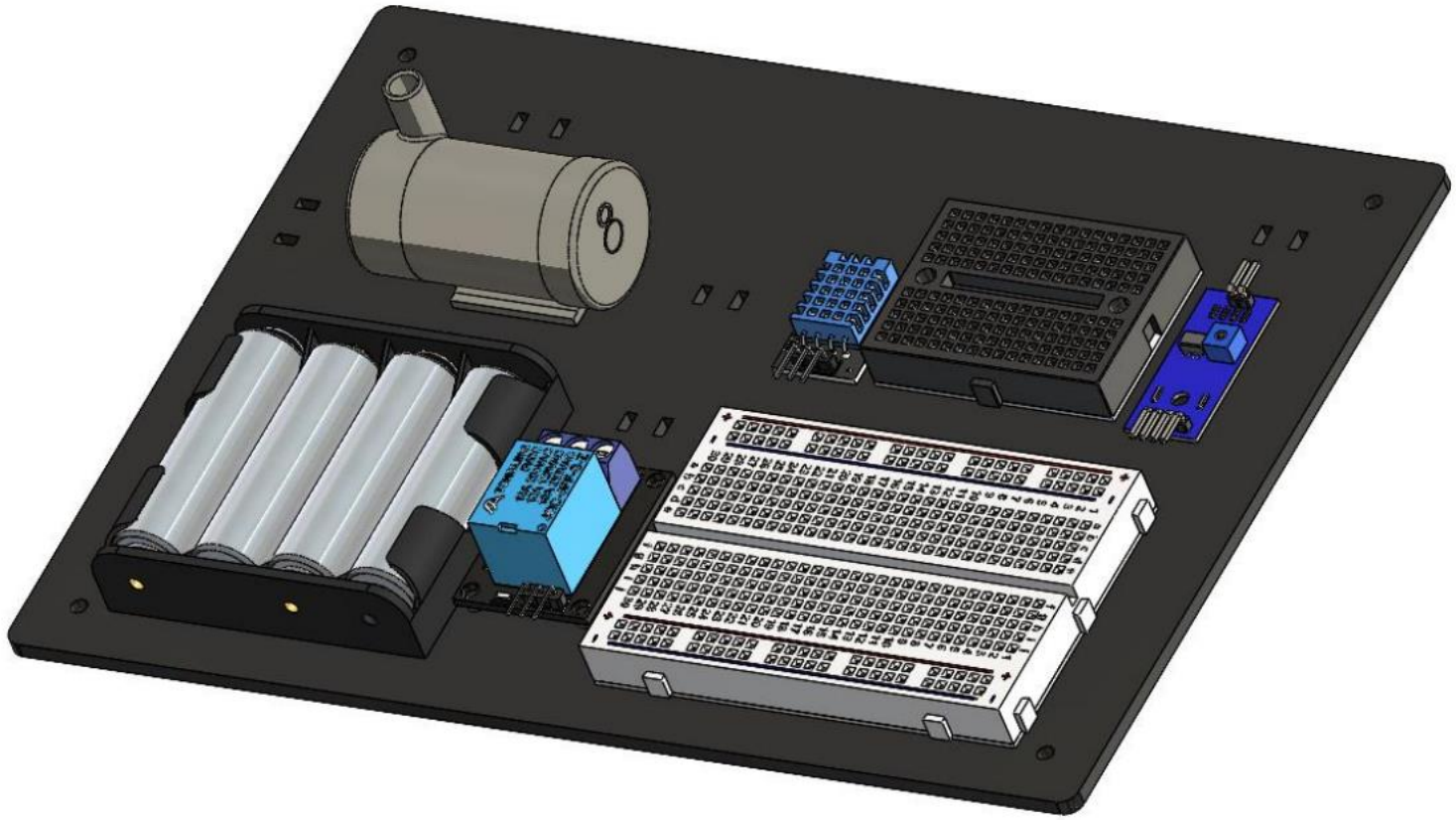
CAD Design

Plan View:

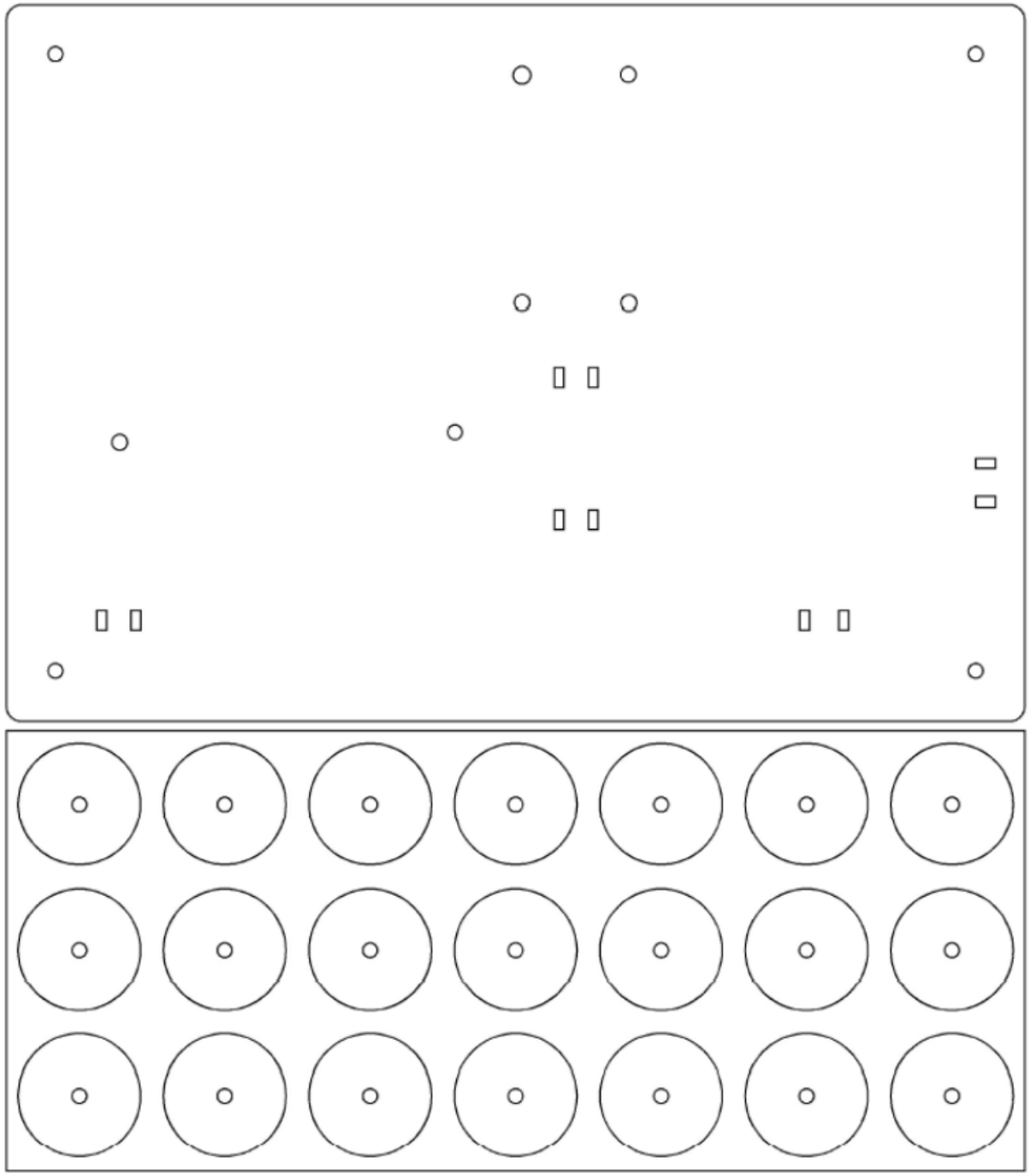


CAD Design

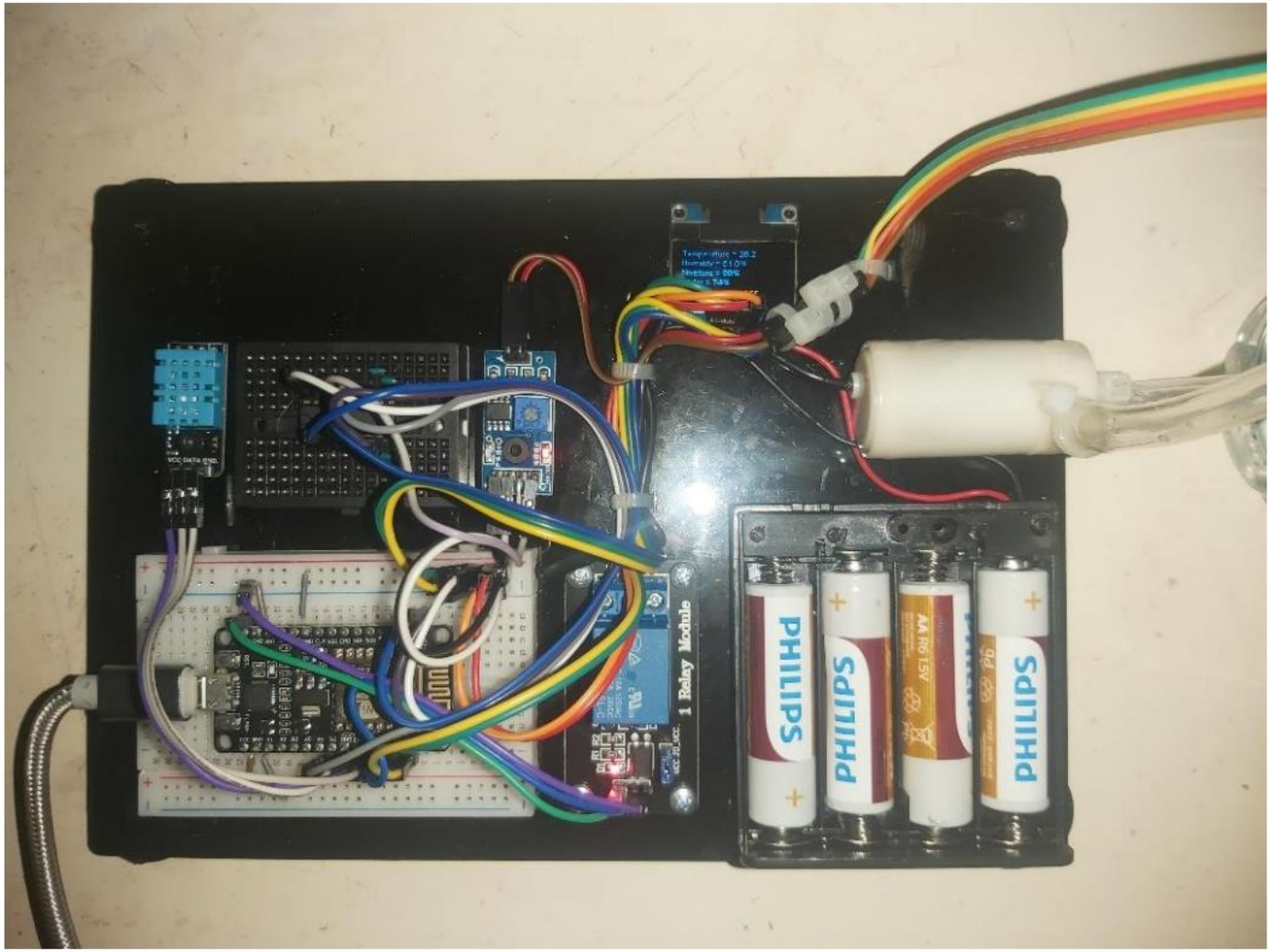
Inclined View:



Laser Cut



Assembly of Complete System



Assembly of Complete System

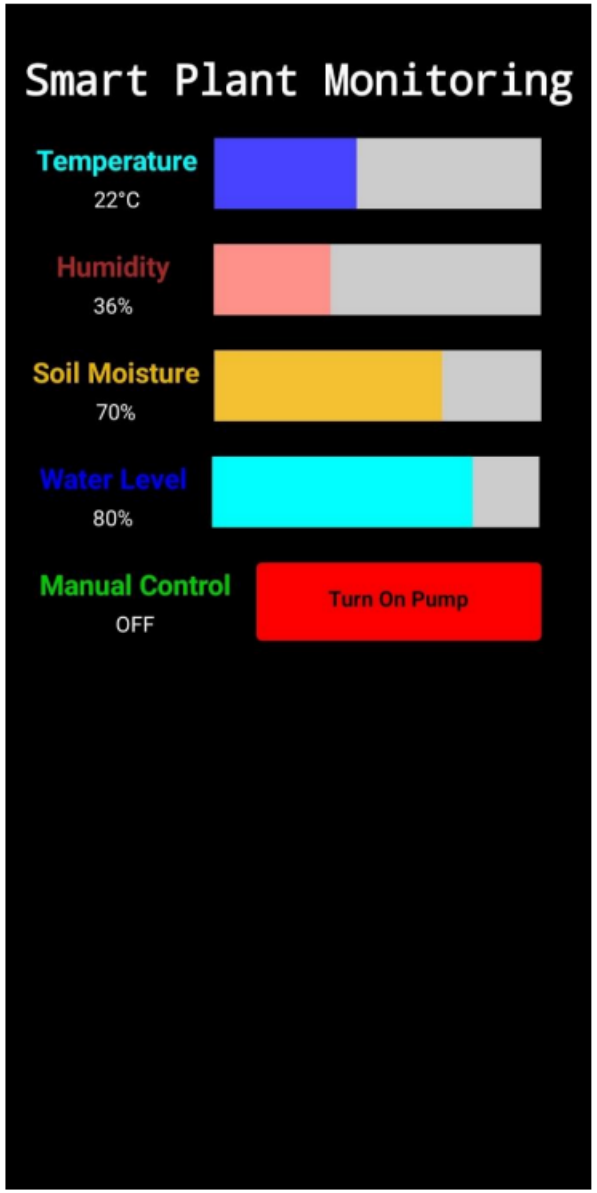
Assembly of Complete System

Mobile Application

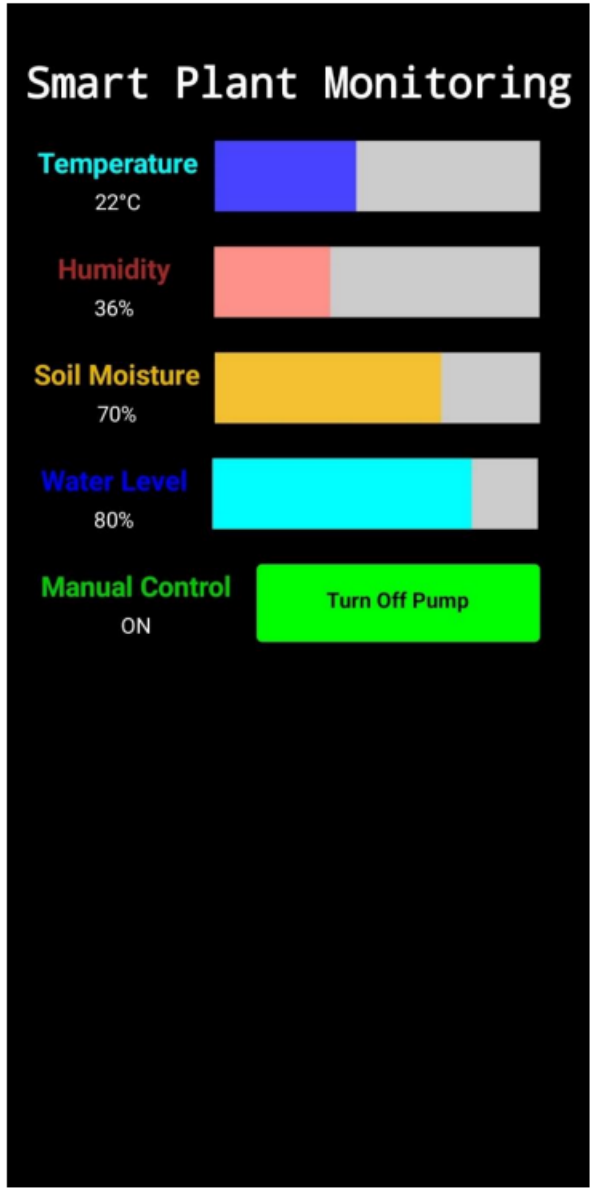
This mobile application is created by MIT App Inventor.

This mobile application reads the temperature, humidity, soil moisture, and water level sensor from Firebase. Then, update the progress bars depending on the values retrieved from Firebase.

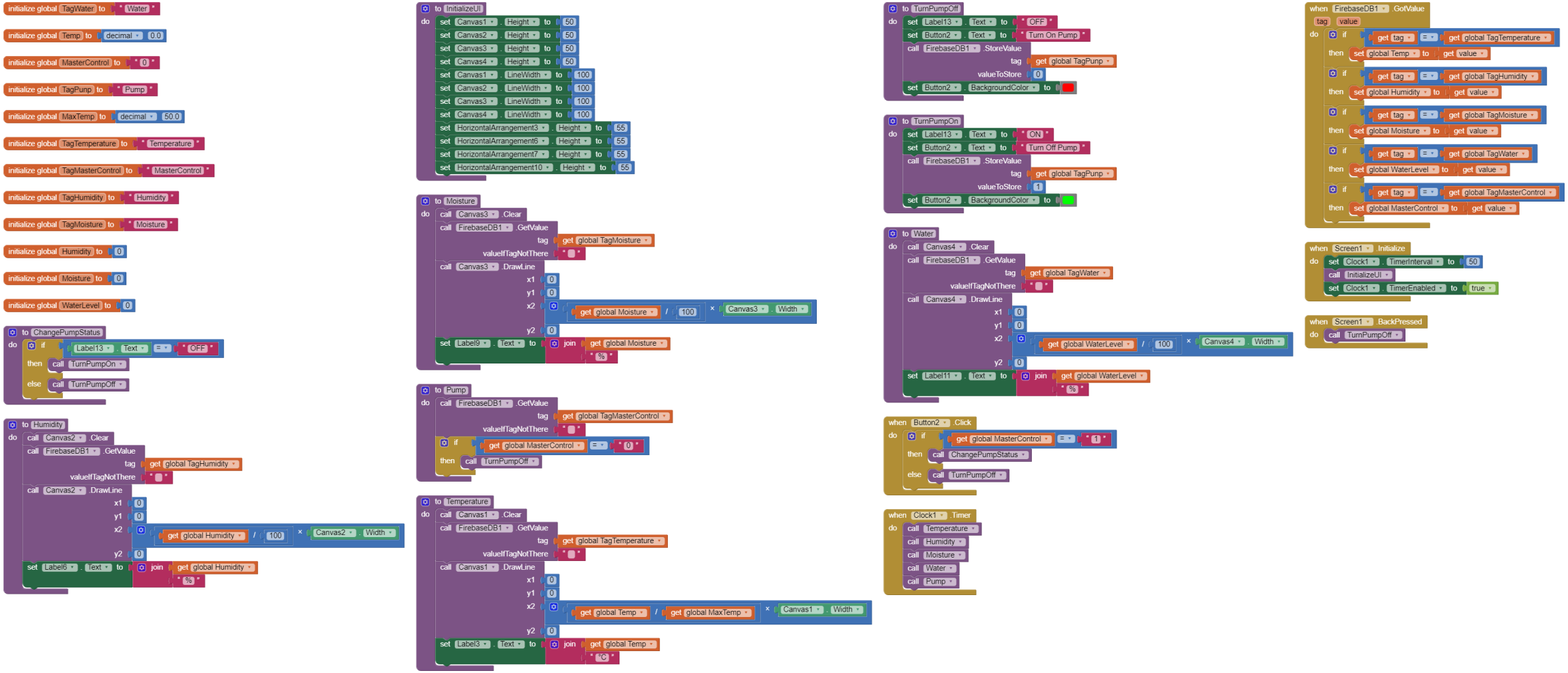
Manual Control is OFF:



Manual Control is ON:

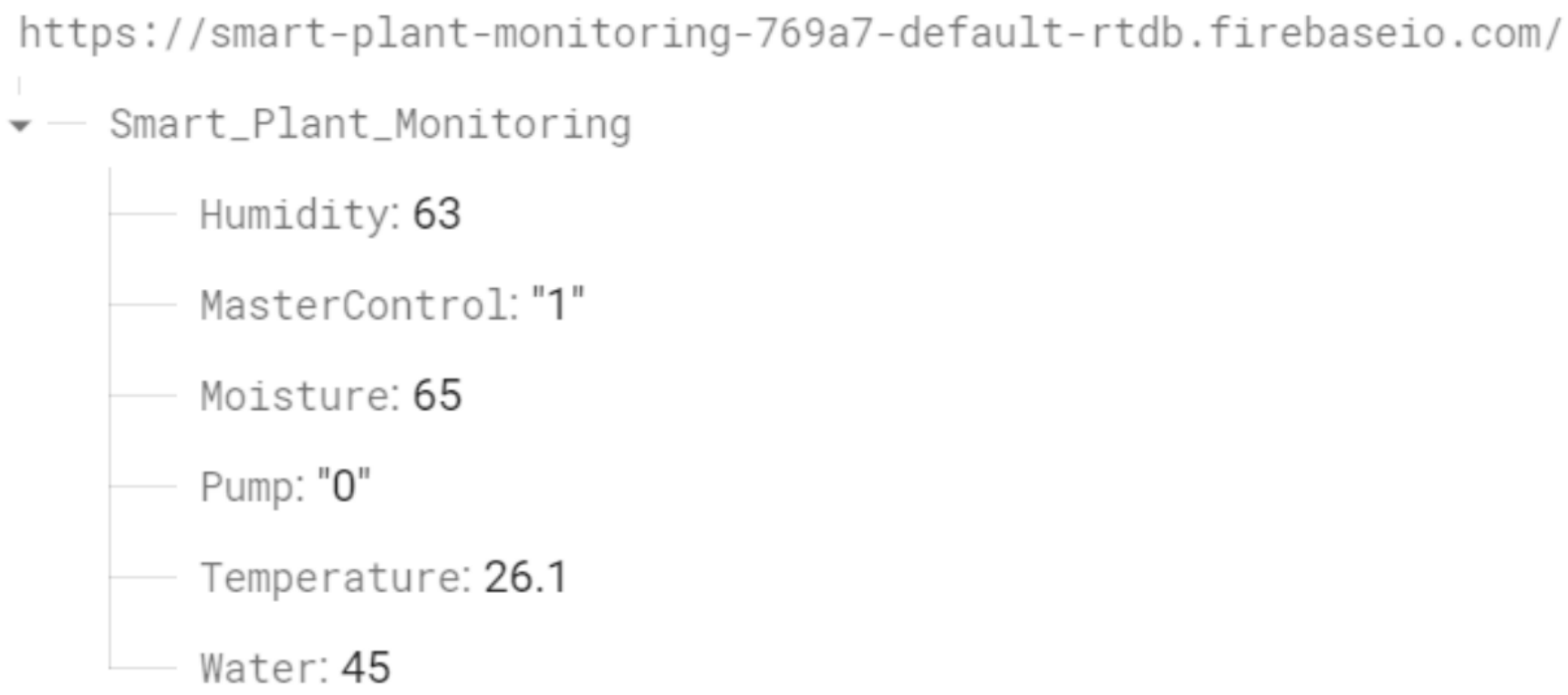


Mobile Application: Blocks



Firebase

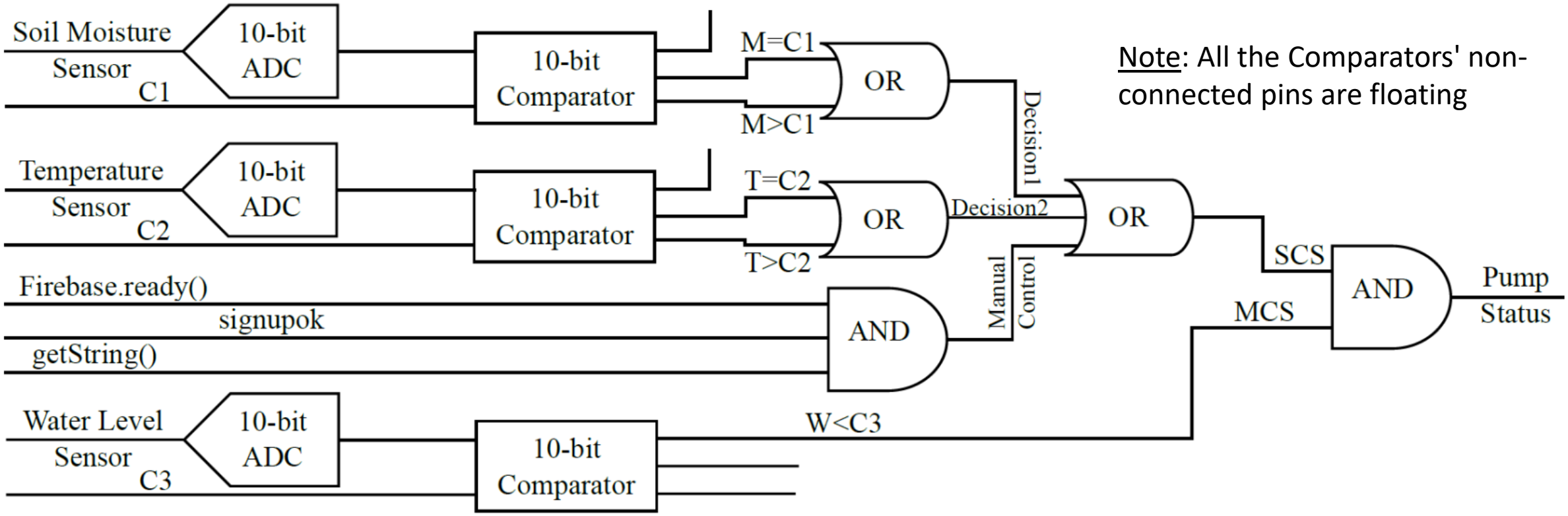
https://smart-plant-monitoring-769a7-default-rtdb.firebaseio.com



Firebase & Mobile Application

Also, the mobile application, can turn ON the pump by clicking on the button “turn on pump” if the MasterControl is high. The MasterControl is high when the water level is not high. If the water level is high, the MasterControl will be low, and this will turn OFF the pump even if it is turned ON by the user, and the status of button in mobile application will be changed accordingly.

Logic Circuit Diagram



Note: All the Comparators' non-connected pins are floating

- SCS: Slave Control Signal - MCS: Master Control Signal - M: Moisture Level
- C1: Moisture_Dry = 750_{10} = 10 1110 1110_2 - T: Temperature in degree Celsius
- W: Water Level - C3: Water_High = 600_{10} = 10 0101 1000_2
- C2: TempHigh = 30°C ≈ 614_{10} = 10 0110 0110_2, this is because the temperature range that DHT11 can measure is between 0 and 50°C.

Code

The program consists of 3 files:

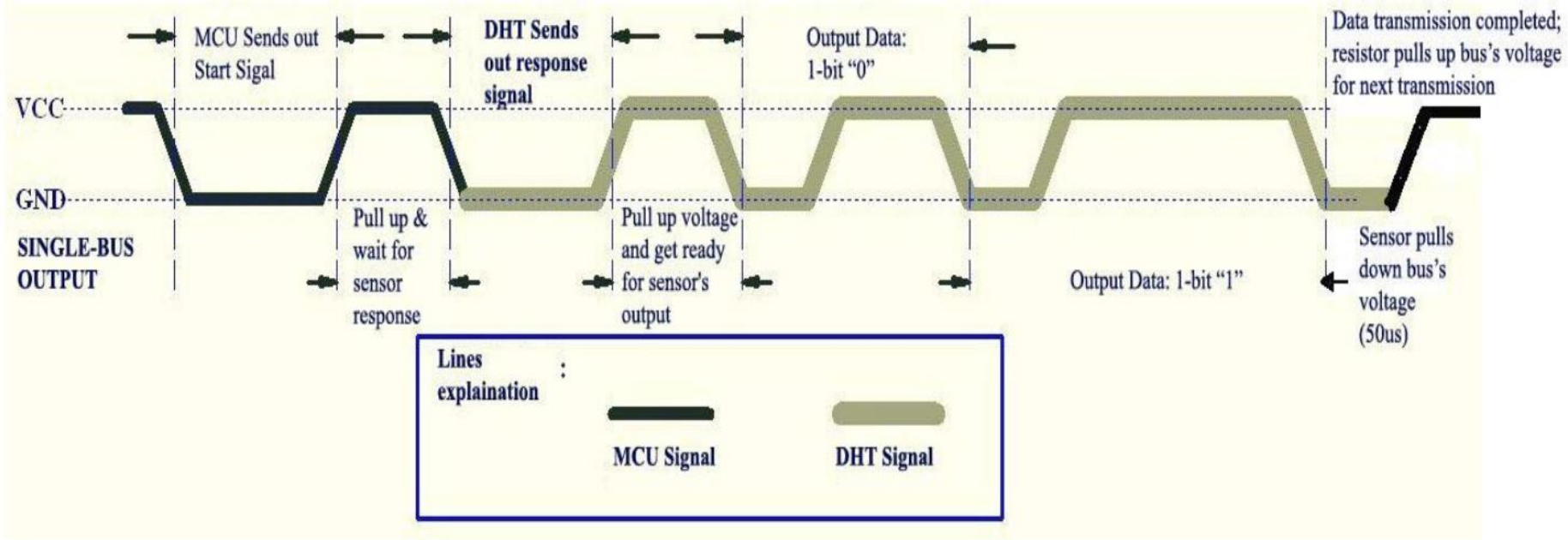
- Code.ino (main file)
- helper_functions.h (contains the declarations of functions and variables)
- helper_functions.cpp (contains the implementation of functions).

```
void setup() {
  Serial.begin(SerialBaudRate);
  initWiFi();
  initFirebase();
  initDHTSensor();
  initAnalogSensors();
  initPump();
  initDisplay();
  updateScreen();
}
```

```
void loop() {
  Serial.println("********** New Loop **********");
  Decision2 = getDHTData();
  Decision1 = getMoistureData();
  delay(100);
  MasterControl = getWaterData();
  delay(1000);
  controlPunp();
  updateFirebase();
  updateScreen();
}
```

How DHT11 sensor works?

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the lowpower-consumption mode until it receives a start signal from MCU again.



Initially – Before Response

- Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000]
- Cycles = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
- lowCycles = highCycles = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Just after one response

- Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000]
- Cycles = [7,3,7,3,7,11,7,11,7,10,8,3,7,10,8,10,8,3,7,3,7,2,8,3,7,3,7,3,8,3,7,3,7,3,8,2,8,3,7,11,7,10,8,3,7,10,7,10,8,3,7,3,7,3,8,3,7,3,7,11,7,11,7,10,8,3,6,10,8,3,7,11,7,10,8,10,8,3,7,10]
- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]

Updating Data: iteration#: 0

- lowCycles = [**7**,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [**3**,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [0,0,0,0,0]

- If highCycles[0] > lowCycles[0]: Data[0] | 1

Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [0,0,0,0,0]

Updating Data: iteration#: 1

- lowCycles = [7,**7**,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,**3**,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [0,0,0,0,0]

- If highCycles[1] > lowCycles[1]: Data[0] | 1

Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [0,0,0,0,0]

Updating Data: iteration#: 2

- lowCycles = [7,7,**7**,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,**11**,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 0000, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [0,0,0,0,0]

- If highCycles[2] > lowCycles[2]: Data[0] | 1

Data = [0000 0001, 00000000, 00000000, 00000000, 00000000] = [1,0,0,0,0]

Updating Data: iteration#: 3

- lowCycles = [7,7,7,**7**,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,**11**,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 0010, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [2,0,0,0,0]

- If highCycles[3] > lowCycles[3]: Data[0] | 1

Data = [0000 0011, 00000000, 00000000, 00000000, 00000000] = [3,0,0,0,0]

Updating Data: iteration#: 4

- lowCycles = [7,7,7,7,**7**,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,**10**,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 0110, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [6,0,0,0,0]

- If highCycles[4] > lowCycles[4]: Data[0] | 1

Data = [0000 0111, 00000000, 00000000, 00000000, 00000000] = [7,0,0,0,0]

Updating Data: iteration#: 5

- lowCycles = [7,7,7,7,7,**8**,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,**3**,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0000 1110, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [14,0,0,0,0]

- If highCycles[5] > lowCycles[5]: Data[0] | 1

Data = [0000 1110, 00000000, 00000000, 00000000, 00000000] = [14,0,0,0,0]

Updating Data: iteration#: 6

- lowCycles = [7,7,7,7,7,8,**7**,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,**10**,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0001 1100, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [28,0,0,0,0]

- If highCycles[6] > lowCycles[6]: Data[0] | 1

Data = [0001 1101, 00000000, 00000000, 00000000, 00000000] = [29,0,0,0,0]

Updating Data: iteration#: 7 (last one for element 1 in Data)

- lowCycles = [7,7,7,7,7,8,7,**8**,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,**10**,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- First element of Data will be updated as follow:

- Data[0] << 1

Data = [0011 1010, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [58,0,0,0,0]

- If highCycles[7] > lowCycles[7]: Data[0] | 1

Data = [0011 1011, 00000000, 00000000, 00000000, 00000000] = [59,0,0,0,0]

Updating Data: iteration#: 8

- lowCycles = [7,7,7,7,7,8,7,8,**8**,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,**3**,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[8] > lowCycles[8]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 9

- lowCycles = [7,7,7,7,7,8,7,8,8,**7**,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,**3**,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[9] > lowCycles[9]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 10

- lowCycles = [7,7,7,7,7,8,7,8,8,7,**7**,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,**2**,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[10] > lowCycles[10]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 11

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,**8**,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,**3**,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[11] > lowCycles[11]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 12

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,**7**,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,**3**,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[12] > lowCycles[12]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 13

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,**7**,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,**3**,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[13] > lowCycles[13]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 14

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,**8**,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,**3**,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[14] > lowCycles[14]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 15 (last one for element 2 in Data)

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,**7**,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,**3**,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Second element of Data will be updated as follow:

- Data[1] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[15] > lowCycles[15]: Data[1] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 16

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,**7**,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,**3**,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[16] > lowCycles[16]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 17

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,**8**,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,**2**,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[17] > lowCycles[17]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 18

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,**8**,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,**3**,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[18] > lowCycles[18]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

Updating Data: iteration#: 19

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,**7**,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,**11**,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0000, 0000 0000, 0000 0000] = [59,0,0,0,0]

- If highCycles[19] > lowCycles[19]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0001, 0000 0000, 0000 0000] = [59,0,1,0,0]

Updating Data: iteration#: 20

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,**7**,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,**10**,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0010, 0000 0000, 0000 0000] = [59,0,2,0,0]

- If highCycles[20] > lowCycles[20]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0011, 0000 0000, 0000 0000] = [59,0,3,0,0]

Updating Data: iteration#: 21

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,**8**,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,**3**,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 0110, 0000 0000, 0000 0000] = [59,0,6,0,0]

- If highCycles[21] > lowCycles[21]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 0110, 0000 0000, 0000 0000] = [59,0,6,0,0]

Updating Data: iteration#: 22

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,**7**,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,**10**,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0000 1100, 0000 0000, 0000 0000] = [59,0,12,0,0]

- If highCycles[22] > lowCycles[22]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0000 1101, 0000 0000, 0000 0000] = [59,0,13,0,0]

Updating Data: iteration#: 23 (last one for element 3 in Data)

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,**7**,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,**10**,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Third element of Data will be updated as follow:

- Data[2] << 1

Data = [0011 1011, 0000 0000, 0001 1010, 0000 0000, 0000 0000] = [59,0,26,0,0]

- If highCycles[23] > lowCycles[23]: Data[2] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 24

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,**8**,7,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,**3**,3,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[24] > lowCycles[24]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 25

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,**7**,7,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,**3**,3,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[25] > lowCycles[25]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 26

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,**7**,8,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,**3**,3,3,11,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[26] > lowCycles[26]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 27

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,**8**,7,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,**3**,3,11,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[27] > lowCycles[27]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 28

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,**7**,7,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,**3**,11,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[28] > lowCycles[28]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

Updating Data: iteration#: 29

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,**7**,7,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,**11**,11,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0000, 0000 0000] = [59,0,27,0,0]

- If highCycles[29] > lowCycles[29]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0001, 0000 0000] = [59,0,27,1,0]

Updating Data: iteration#: 30

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,**7**,7,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,**11**,10,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0010, 0000 0000] = [59,0,27,2,0]

- If highCycles[30] > lowCycles[30]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0011, 0000 0000] = [59,0,27,3,0]

Updating Data: iteration#: 31 (last one for element 4 in Data)

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,**7**,8,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,**10**,3,10,3,11,10,10,3,10]
- Fourth element of Data will be updated as follow:

- Data[3] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0110, 0000 0000] = [59,0,27,6,0]

- If highCycles[31] > lowCycles[31]: Data[3] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0000] = [59,0,27,7,0]

Updating Data: iteration#: 32

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,**8**,6,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,**3**,10,3,11,10,10,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0000] = [59,0,27,7,0]

- If highCycles[32] > lowCycles[32]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0000] = [59,0,27,7,0]

Updating Data: iteration#: 33

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,**6**,8,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,**10**,3,11,10,10,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0000] = [59,0,27,7,0]

- If highCycles[33] > lowCycles[33]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0001] = [59,0,27,7,1]

Updating Data: iteration#: 34

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,**8**,7,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,**3**,11,10,10,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0010] = [59,0,27,7,2]

- If highCycles[34] > lowCycles[34]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0010] = [59,0,27,7,2]

Updating Data: iteration#: 35

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,**7**,7,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,**11**,10,10,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0100] = [59,0,27,7,4]

- If highCycles[35] > lowCycles[35]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 0101] = [59,0,27,7,5]

Updating Data: iteration#: 36

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,**7**,8,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,**10**,10,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 1010] = [59,0,27,7,10]

- If highCycles[36] > lowCycles[36]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0000 1011] = [59,0,27,7,11]

Updating Data: iteration#: 37

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,**8**,8,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,**10**,3,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0001 0110] = [59,0,27,7,22]

- If highCycles[37] > lowCycles[37]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0001 0111] = [59,0,27,7,23]

Updating Data: iteration#: 38

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,**8**,7]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,**3**,10]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0010 1110] = [59,0,27,7,46]

- If highCycles[38] > lowCycles[38]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0010 1110] = [59,0,27,7,46]

Updating Data: iteration#: 39 (last one for element 5 in Data)

- lowCycles = [7,7,7,7,7,8,7,8,8,7,7,8,7,7,8,7,7,8,8,7,7,8,7,7,8,7,7,8,7,7,7,7,8,6,8,7,7,8,8,**7**]
- highCycles = [3,3,11,11,10,3,10,10,3,3,2,3,3,3,3,3,3,2,3,11,10,3,10,10,3,3,3,3,3,11,11,10,3,10,3,11,10,10,3,**10**]
- Fifth element of Data will be updated as follow:

- Data[4] << 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0101 1100] = [59,0,27,7,92]

- If highCycles[39] > lowCycles[39]: Data[4] | 1

Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0101 1101] = [59,0,27,7,93]

Data Validation

- The received data is considered valid if their sum equals the fifth segment (the checksum).
- Data = [0011 1011, 0000 0000, 0001 1011, 0000 0111, 0101 1101] = [59,0,27,7,93]
- Data[0] + Data[1] + Data[2] + Data[3] =

$$\begin{array}{rcccccccc} & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ + & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ + & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ + & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

- Data[0] + Data[1] + Data[2] + Data[3] = 0101 1101

Bitwise Masking:

- Mask = 0xFF = 1111 1111
- Data[0] + Data[1] + Data[2] + Data[3] = 0101 1101
- Sum = (Data[0] + Data[1] + Data[2] + Data[3]) &0xFF:

$$\begin{array}{rcccccccc} & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ \& & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

- Checksum = 0101 1101
- Is checksum = Sum? **Yes**

$\therefore$ Data might be valid

H = Data[0] + 0.1×Data[1] = 59+0.1x0 = 59%

T = Data[2] + 0.1×Data[3] = 27+0.1x7 = 27.7°C

Generating All Possible Test Cases Scenarios

The main objective of generating all possible test case scenarios is to ensure that all cases are met and none are missed.

Possible Values For Each Parameter:

```
WaterLevel = {"0-599", "600-1024"}
C1 = "0-599" #<600
Firebase_ready = {"0", "1"}
signupok = {"0", "1"}
getString = {"0", "1"}
SoilMoistureSensor = {"0-749", "750-1024"}
C2 = "0-749" #<750
TemperatureSensor = {"0-29", "30-50"}
C3 = "0-29" #<30
userInput = {"0", "1"}
```

Generating All Possible Test Cases Scenarios: Sample Output

	Water Level	Firebase_ready	signupok	getString	SoilMoistureSensor	TemperatureSensor	userInput	MasterControlSignal	Decision1	Decision2	ManualControl	SlaveControlSignal	PumpStatus
7	0-599	1	1	1	750-1024	0-29	0	1	1	0	0	1	1
8	0-599	1	1	0	0-749	30-50	1	1	0	1	0	1	1
9	0-599	1	1	0	0-749	30-50	0	1	0	1	0	1	1
10	0-599	1	1	0	0-749	0-29	1	1	0	0	0	0	0
11	0-599	1	1	0	0-749	0-29	0	1	0	0	0	0	0
12	0-599	1	1	0	750-1024	30-50	1	1	1	1	0	1	1
13	0-599	1	1	0	750-1024	30-50	0	1	1	1	0	1	1
14	0-599	1	1	0	750-1024	0-29	1	1	1	0	0	1	1
15	0-599	1	1	0	750-1024	0-29	0	1	1	0	0	1	1
16	0-599	1	0	1	0-749	30-50	1	1	0	1	0	1	1
17	0-599	1	0	1	0-749	30-50	0	1	0	1	0	1	1
18	0-599	1	0	1	0-749	0-29	1	1	0	0	0	0	0

Verifying Generated Test Cases

```
Test Case 1: Passed     Test Case 2: Passed     Test Case 3: Passed     Test Case 4: Passed
Test Case 5: Passed     Test Case 6: Passed     Test Case 7: Passed     Test Case 8: Passed
Test Case 9: Passed     Test Case 10: Passed    Test Case 11: Passed    Test Case 12: Passed
Test Case 13: Passed    Test Case 14: Passed    Test Case 15: Passed    Test Case 16: Passed
Test Case 17: Passed    Test Case 18: Passed    Test Case 19: Passed    Test Case 20: Passed
Test Case 21: Passed    Test Case 22: Passed    Test Case 23: Passed    Test Case 24: Passed
Test Case 25: Passed    Test Case 26: Passed    Test Case 27: Passed    Test Case 28: Passed
Test Case 29: Passed    Test Case 30: Passed    Test Case 31: Passed    Test Case 32: Passed
Test Case 33: Passed    Test Case 34: Passed    Test Case 35: Passed    Test Case 36: Passed
Test Case 37: Passed    Test Case 38: Passed    Test Case 39: Passed    Test Case 40: Passed
Test Case 41: Passed    Test Case 42: Passed    Test Case 43: Passed    Test Case 44: Passed
Test Case 45: Passed    Test Case 46: Passed    Test Case 47: Passed    Test Case 48: Passed
Test Case 49: Passed    Test Case 50: Passed    Test Case 51: Passed    Test Case 52: Passed
Test Case 53: Passed    Test Case 54: Passed    Test Case 55: Passed    Test Case 56: Passed
Test Case 57: Passed    Test Case 58: Passed    Test Case 59: Passed    Test Case 60: Passed
Test Case 61: Passed    Test Case 62: Passed    Test Case 63: Passed    Test Case 64: Passed
Test Case 65: Passed    Test Case 66: Passed    Test Case 67: Passed    Test Case 68: Passed
Test Case 69: Passed    Test Case 70: Passed    Test Case 71: Passed    Test Case 72: Passed
Test Case 73: Passed    Test Case 74: Passed    Test Case 75: Passed    Test Case 76: Passed
Test Case 77: Passed    Test Case 78: Passed    Test Case 79: Passed    Test Case 80: Passed
Test Case 81: Passed    Test Case 82: Passed    Test Case 83: Passed    Test Case 84: Passed
Test Case 85: Passed    Test Case 86: Passed    Test Case 87: Passed    Test Case 88: Passed
Test Case 89: Passed    Test Case 90: Passed    Test Case 91: Passed    Test Case 92: Passed
Test Case 93: Passed    Test Case 94: Passed    Test Case 95: Passed    Test Case 96: Passed
Test Case 97: Passed    Test Case 98: Passed    Test Case 99: Passed    Test Case 100: Passed
Test Case 101: Passed   Test Case 102: Passed   Test Case 103: Passed   Test Case 104: Passed
Test Case 105: Passed   Test Case 106: Passed   Test Case 107: Passed   Test Case 108: Passed
Test Case 109: Passed   Test Case 110: Passed   Test Case 111: Passed   Test Case 112: Passed
Test Case 113: Passed   Test Case 114: Passed   Test Case 115: Passed   Test Case 116: Passed
Test Case 117: Passed   Test Case 118: Passed   Test Case 119: Passed   Test Case 120: Passed
Test Case 121: Passed   Test Case 122: Passed   Test Case 123: Passed   Test Case 124: Passed
Test Case 125: Passed   Test Case 126: Passed   Test Case 127: Passed   Test Case 128: Passed

                        Passed Test Cases = 128
```

References

[1] Texas Instruments. (2021). *CD74HC4051-EP Analog Multiplexer and Demultiplexer.* Retrieved from https://www.ti.com/lit/ds/symlink/cd74hc4051-ep.pdf

[2] Texas Instruments. (2015). *Understanding the I2C Bus*. Retrieved from https://www.ti.com/lit/an/slva704/slva704.pdf

[3] Espressif Systems. (2023). *ESP8266EX Datasheet*. Retrieved from https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[4] Arduino. (2009). *Arduino Uno Datasheet*. Retrieved from https://docs.arduino.cc/resources/datasheets/A000066- datasheet.pdf

[5] Mouser Electronics. *DHT11 Humidity & Temperature Sensor*. Retrieved from https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[6] https://appinventor.mit.edu/

Thank You