**Methodology of FrenziTech IoT Device**

**Explanation of Pin Utilization of ESP32 ucontroller**

ESP32 DevKitC-V4 has total of 38 pins. It has x3 GND pins for ground, x1 5V for 5volts power source, x1 3V3 pin for 3.3 volts power source and an EN pin for enable purpose. We can use them multiple times but it is better to provide external power source to power sensors and actuators. Therefore excluding 6 above pins there are a total of 32 GPIO pins.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **G21** | **RxD** | **TxD** | **G22** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 1: 5V, 3V3 in **RED**, GND in **BLACK** and EN in **GREEN**

x6 GPIOs excluding 3V3 and GND pins are being used from a uController ESP32 to LoRa Ra02 for communication purpose. These pins are ***G02***->DIO0, ***G14***->RES, ***G05***->NSS, ***G18***->SCK, ***G23***->MOSI and ***G19***->MISO from ESP32 -> LoRa Ra02 device. Excluding these 6 pins now 26 GPIOs are left.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **G21** | **RxD** | **TxD** | **G22** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 2: G02, G14, G05, G18, G19 G23 in **BLUE** connected to LoRa

***VP/G36 and VN/G39*** are digital pins and have no capability of PWM function therefore using these x2 pins as digital IN.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **G21** | **RxD** | **TxD** | **G22** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 3: G35 and G34, in **GREEN** connected for Digital IN pins.

There are x16 ADC pins which can be used as an analog to digital converter, only few analog pins are required therefore using x6 GPIOs for analog IN function to measure 0-3.3 volt signals, therefore utilizing ***G34 (ADC1\_6), G35(ADC1\_7), G32(ADC1\_4), G33(ADC1\_5) , G25(ADC1\_8) and G26(ADC2\_9)*** GPIO pins.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **G21** | **RxD** | **TxD** | **G22** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 4: G34, G35, G32, G33, G25 and G26 in **PURPLE** connected for Analog IN pins.

In field we also need to control the actuators therefor using ***G27, G12, G13, G15, G22 and G21*** x6GPIOs for this purpose. These pins also have a PWM generation capability therefore we can control the actuators output voltages from 0 to 255.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **SDA** | **RxD** | **TxD** | **SCL** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 4: G27 and G12 in **Orange** connected for Digital OUT pins.

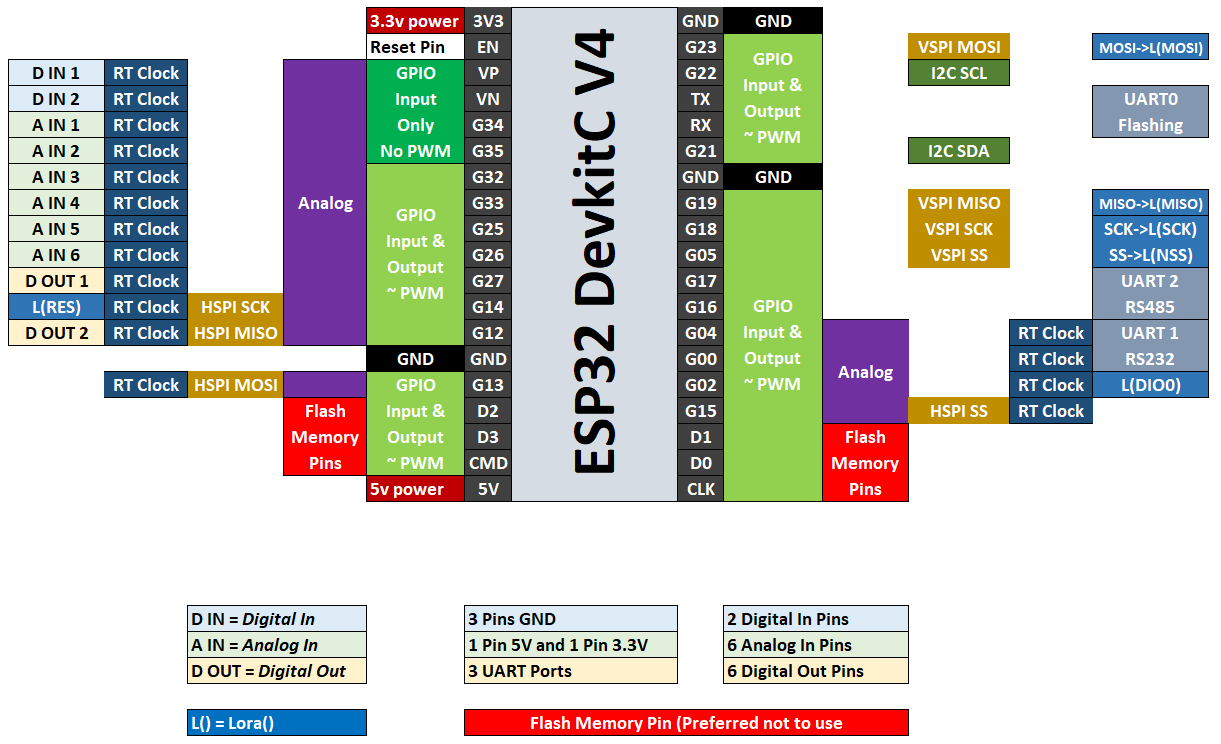
There are x3 serial ports. Two of them port 0 and port 2 can be used without reinitialization whereas port 1 needs reinitialization. Port 0 is ***Rx/TxD***, Port 2 is ***G16/G17*** and Port 1 that needs reinitialization is ***G0/G04***. Port 2 is used for RS232 communication whereas Port 1 is used for RS485 communication and Port 0 is used for programming purpose.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5V** | **D0/G07** | **D1/G08** | **G15** | **G02** | **G00** | **G04** | **G16** | **G17** | **G05** | **G18** | **G19** | **GND** | **SDA** | **RxD** | **TxD** | **SCL** | **G23** | **GND** |
| **ESP32 DevKitC – V4** | | | | | | | | | | | | | | | | | | |
| **CLK/G06** | **CMD/G11** | **D3/G10** | **D2/G09** | **G13** | **GND** | **G12** | **G14** | **G27** | **G26** | **G25** | **G33** | **G32** | **G35** | **G34** | **VN/G39** | **VP/G36** | **EN** | **3V3** |

Table 5: RxD, TxD, G16, G17, G00 and G04 in **Gray** connected for serial communication purposes.

X6 pins are left out which will be used further developing the code. Issue regarding not utilizing them now is that they are connected to memory.

**Vertical Table of Pin Utilization Diagram**



**MAIN BOARD SPEC**

|  |  |  |
| --- | --- | --- |
| INPUT AC VOLTAGE | 110-240V AC |  |
| INPUT DC VOLTAGE | 5V OR SOLAR |  |
| BATTERY BACKUP | 18305 LI-ION |  |
| CONNECTIONS | WIFI, LORA, BT, LTE |  |
| INTERNAL VOLTAGES | 3.3V, 5V, 12V, 18V, 24V? |  |
| DIGITAL IOS | 2-RELAYS 5V – 12 - 24V | EXTENDED TO 256 |
| ACTUATORS | 2-SOLENOIDS |  |
| PROTOCOLS | RS232, I2C, SPI, RS485 |  |
| ANALOG IN | 6 – BAT IN VOLT, BOARD TEMP, MOTOR CURRENT, 4-20mA, 5V IN, 3.3V |  |
| SENSORS | TEMP, MOISTURE, SALINITY, CURRENT, WELL DEPTH |  |
| MASTER | ESP32 MCU |  |
| SLAVE | Arduino |  |

**REMOTE BOARD SPEC**

|  |  |  |
| --- | --- | --- |
| INPUT AC VOLTAGE | 110-240V AC |  |
| INPUT DC VOLTAGE | 5V OR SOLAR |  |
| BATTERY BACKUP | 18305 LI-ION |  |
| CONNECTIONS | LORA |  |
| INTERNAL VOLTAGES | 3.3V, 5V, 12V, 18V, 24V? |  |
| DIGITAL IOS | 2-RELAYS 5V – 12 - 24V | EXTENDED TO 256 |
| ACTUATORS | 2-SOLENOIDS |  |
| PROTOCOLS | RS232, I2C, SPI, RS485 |  |
| ANALOG IN | 6 – BAT IN VOLT, BOARD TEMP, MOTOR CURRENT, 4-20mA, 5V IN, 3.3V |  |
| SENSORS | TEMP, MOISTURE, SALINITY, CURRENT, WELL DEPTH |  |
| MASTER | ESP32 MCU |  |
| SLAVE | Arduino, ESP32CAM |  |

**Explanation of Block Diagram**

Suppose there are multiple RTUs and a single MCU. These multiple RTUs are communicating to MCU through LoRa Ra02 using an SPI communication utilizing 6 GPIO pins (G02, G14, G05, G18 and G19).

RTU and MCU has x2 digital IN pins (G36 and G39), x6 analog IN pins (G34, G35, G32, G33, G25 and G26) and x6 digital OUT pins (G27, G12, G13, G15, G22 and G21). There are x3 serial ports, of which port 2 is used for RS232 communication whereas Port 1 is used for RS485 communication and Port 0 is used for programming purpose.

The only difference between RTU and MCU is that MCU is continuously utilizing its port 2 communicating with 4G LTE device. Through this device the data is being published and subscribed to and from the MQTT cloud.

Right now MQTT cloud is an online server created on <https://mqtthq.com/>. But we will create our own offline MQTT server later.

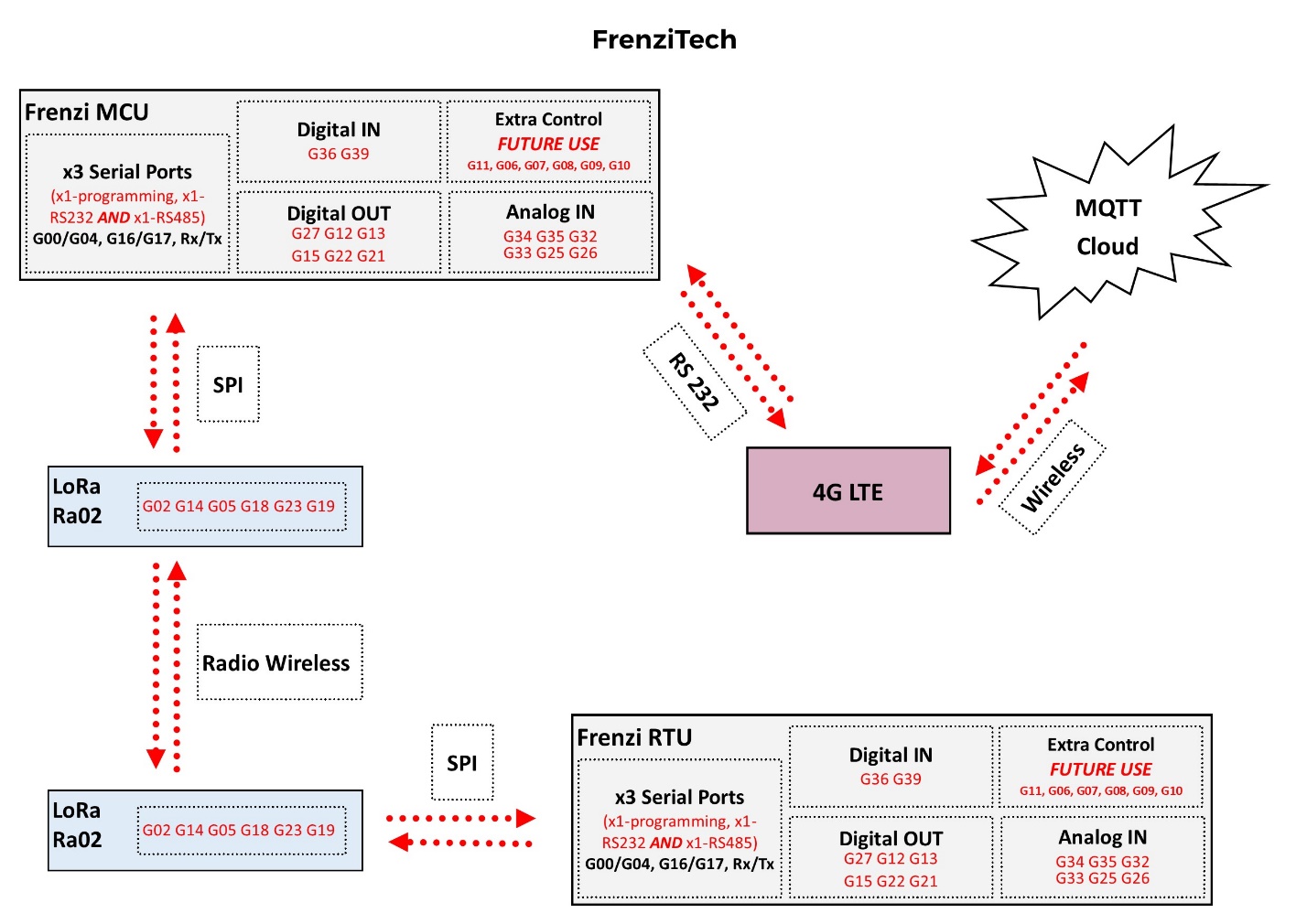


Figure 1: Block Diagram of an IoT device system

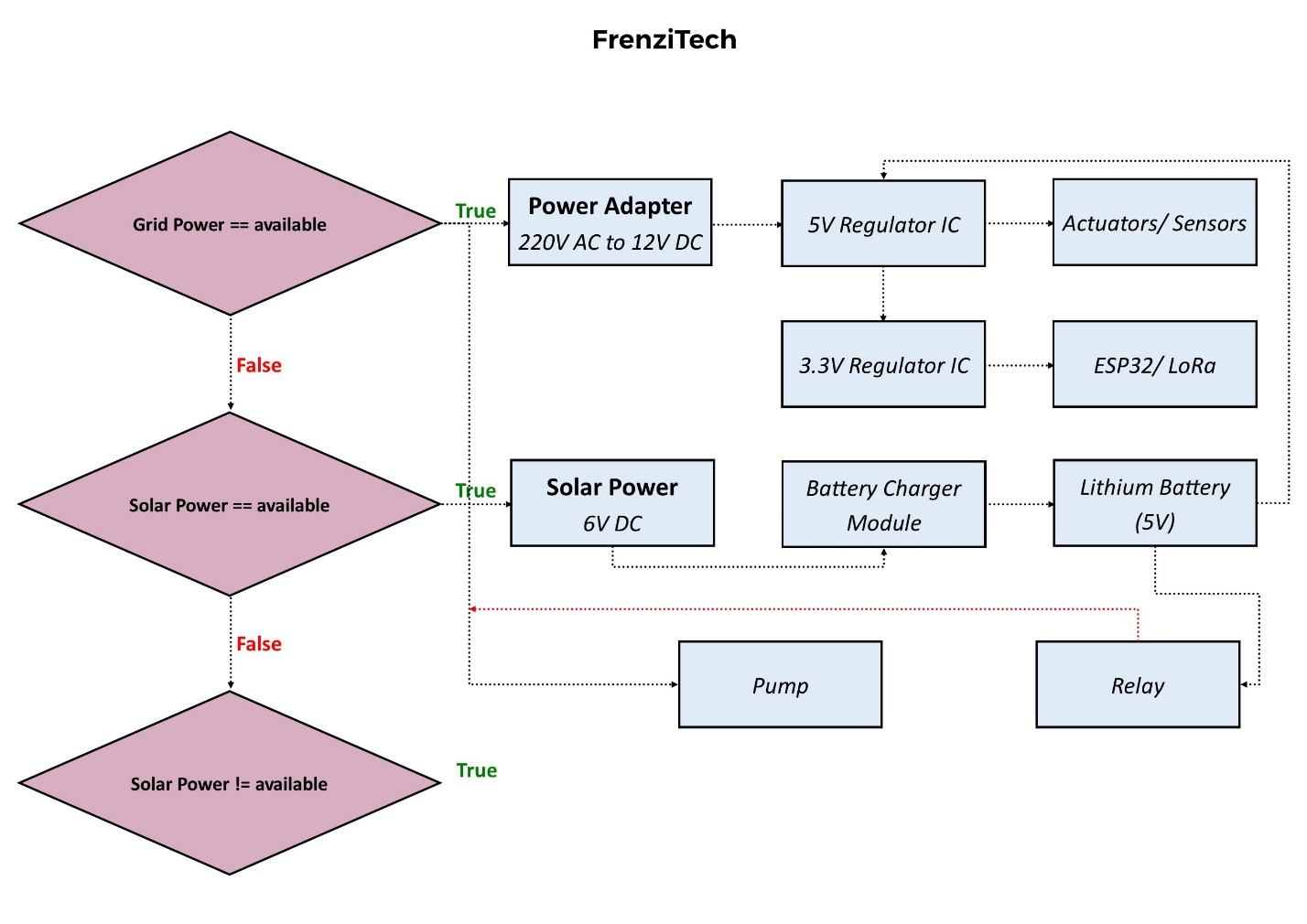


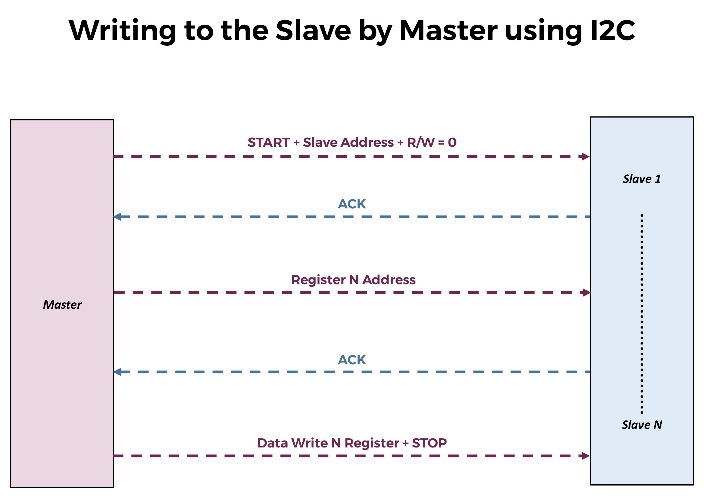
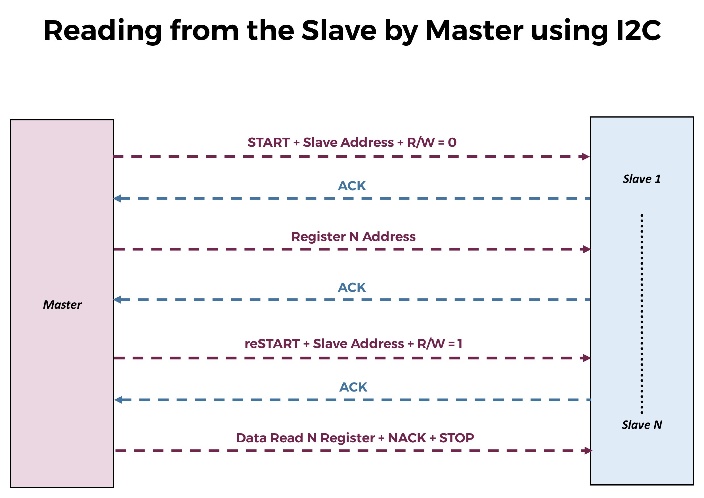
Figure 2: Block diagram for powering the IoT device system

**Explanation of Power System**

The system can be powered on from grid power, solar power and from lithium 5V battery. If actuators are a high voltage devices such as motors and pumps then we will be needing 220v to power them. A relay will be used to control them from ESP32.

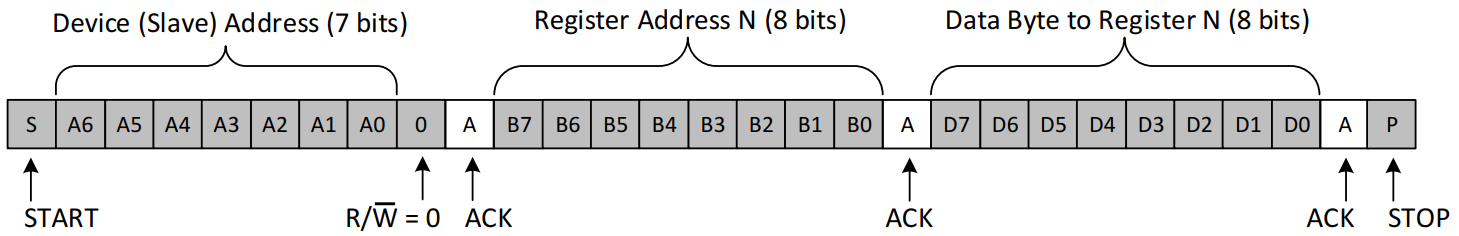
Most of the systems are going to be powered from solar panels source. This solar power source is going to be charged in to 5V battery using battery charger. Then from 5V battery actuators and sensors can be powered on. A 3.3V regulator (LM3940) converts 5V to 3.3V to power the ESP32 and LoRa.

**I2C Communication**

****

**Writing to the Slave Master using I2C**

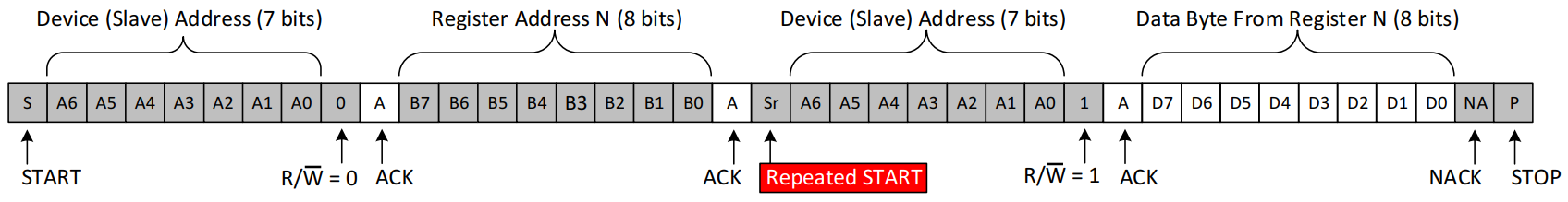
The I2C bus write process involves the master sending a start condition on the bus with the slave's address, with the R/W bit set to 0 indicating a write. Upon receiving the acknowledge bit from the slave, the master sends the register address of the target register it wants to write to. After receiving the acknowledgment from the slave, the master starts sending the register data to the slave. This process continues until all the data required to be written to the register has been sent, which is usually a single byte, and the transmission ends with a STOP condition sent by the master.



**Reading from the Slave Master using I2C**

To read from a slave on the I2C bus, the master starts by instructing the slave which register it wants to read from, following a similar process as writing. The master begins by sending the address with the R/W bit set to 0, followed by the register address it wishes to read from. Once the slave acknowledges the register address, the master sends a START condition again, followed by the slave address with the R/W bit set to 1 indicating a read request. The slave acknowledges the read request, and the master releases the SDA bus but continues to supply the clock to the slave. At this point, the master becomes the master-receiver, and the slave becomes the slave-transmitter, allowing the slave to send the requested data to the master in the subsequent clock cycles.

During the read operation on the I2C bus, the master keeps sending clock pulses while releasing the SDA line so that the slave can transmit data. After receiving each byte of data, the master sends an ACK to the slave, indicating it's ready for more data. Once the master receives the expected number of bytes, it sends a NACK to signal the slave to halt communications and release the bus. The master concludes the transaction by sending a STOP condition.



**I2C communication between ESP32 & Arduino**

**Algorithm ESP32 I2C Master**

1. Include wire library.
2. Define addresses for multiple slaves.
3. In setup function
   1. Begin wire.
   2. Begin serial.
4. In loop function
   1. To write to the I2C slave
      1. Start wire transmission with N slave address.
      2. Wire write to specific 0xNN register.
      3. Wire write data to the above register.
      4. Stop wire transmission.
   2. To read from the I2C slave
      1. Start wire transmission with N slave address
      2. Wire read data from the 0xNN register.
      3. Stop wire transmission.
   3. To serial print the data read from I2C slave
      1. Request the number of bytes to be read from the slave.
      2. If wire communication is available then save the data into a variable and serial print it.
5. Give some delay. (i.e. 1000 milliseconds)

**Algorithm Arduino I2C Slave**

1. Include wire library.
2. Define address for this slave.
3. In setup function
   1. Begin wire with a slave address.
   2. Create an attribute for onReceive() i.e. receiveData.
   3. Create an attribute for onRequest() i.e. requestData.
4. Create receiveData() function with an instance to count bytes.
   1. Read the available bytes and compare if they are equal to same address
   2. If same then copy the read data to a variable.
5. Create requestData() function
   1. Write the data to the variable.
6. Create a loop function
   1. Call the receiveData() function
   2. Give a little delay.
   3. Call the requestData() function
   4. Give a little delay.

**SPI communication between ESP32 & LoRa**

**Algorithm Arduino Sender**

1. Include SPI and LoRa library.
2. Initialize a counter.
3. In setup function
   1. Begin Serial
   2. Set SPI selection, reset and digital I/O pin for LoRa
   3. Begin LoRa at your locations frequency and delay for a half second until it begins.
   4. Set sync word to the same of the receiver.
4. Create a loop function
   1. Begin LoRa packet.
   2. Print data in LoRa.
   3. End LoRa Packet.
   4. Give some delay.

**LoRa Mesh NetWork**

LoRa Mesh is a wireless networking protocol based on the LoRaWAN technology, which allows devices to communicate with each other over a long range, low power wireless network. The main difference between LoRaWAN and LoRa Mesh is that LoRaWAN is a star topology network, whereas LoRa Mesh is a decentralized, mesh topology network.

In a LoRa Mesh network, devices can act as both endpoints and routers, allowing the network to self-organize and self-heal. This means that if one device fails or goes out of range, other devices in the network can still communicate with each other by routing data through intermediate devices.

LoRa Mesh networks are ideal for applications that require long range communication, low power consumption, and high reliability, such as smart cities, industrial automation, and agriculture. They are also well-suited for applications that require real-time monitoring and control, such as asset tracking, environmental sensing, and security systems.

**Reference:**

1. [*nootropicdesign/lora-mesh: LoRa mesh networking (github.com)*](https://github.com/nootropicdesign/lora-mesh)
2. [LoRa Mesh Networking with Simple Arduino-Based Modules | Project Lab (nootropicdesign.com)](https://nootropicdesign.com/projectlab/2018/10/20/lora-mesh-networking/)

**Explanation:**

**-92**

**-97**

**-51**

**-41**

**-90**

**-63**

**-93**

**-73**

**Algorithm**

1. Initialize the LoRa modules on each node with appropriate settings such as frequency, spreading factor, bandwidth, and coding rate.
2. Set up each node to listen for incoming messages and store any incoming messages in a buffer.
3. When a node wants to send a message, it first checks if the destination node is within its radio range. If it is, the message is sent directly to the destination node.
4. If the destination node is not within radio range, the sender broadcasts the message to all nodes within its radio range.
5. When a node receives a broadcast message, it first checks if it is the destination node. If it is, it stores the message in its buffer and sends an acknowledgement message back to the sender.
6. If the node is not the destination node, it checks if it has already received the message before. If it has, it discards the message.
7. If the node has not received the message before, it stores the message in its buffer and rebroadcasts it to all nodes within its radio range except for the sender.
8. Nodes periodically check their buffer for incoming messages and process any messages they find.
9. If a node is unable to communicate with any of its neighbors, it broadcasts a message requesting help to find a new route to the destination node.
10. Nodes receiving a request for help message respond with a message containing the best route they know to the destination node.
11. The requesting node selects the best route and starts using it to send messages to the destination node.
12. Nodes periodically update their routing tables to reflect changes in the network topology.
13. Use a visualization tool to display the network topology in realtime, showing the connections between nodes and any messages being sent or received.

**Basic algorithm for four nodes mesh**

1. Identify the four mesh nodes and assign unique identifiers to each node.
2. Determine the links between the nodes and their weights.
   1. Links can be represented as edges in a graph.
   2. Weights can represent the strength of the connection between nodes.
3. Create an adjacency matrix to represent the links between the nodes. The adjacency matrix should have a size of 4x4.
4. Apply Dijkstra's algorithm to find the shortest path between any two nodes. The algorithm works by finding the node with the lowest distance to the starting node, then updating the distances of all adjacent nodes. Repeat this process until the target node is reached or all nodes have been visited.
5. Use the Bellman-Ford algorithm to find the shortest path between any two nodes, including negative-weight edges. The algorithm works by relaxing edges iteratively, reducing the distance to the target node until convergence.
6. Implement a routing algorithm to determine the best path between nodes. The algorithm can use either Dijkstra's or Bellman-Ford algorithm to find the shortest path. It should take into account the weights of the links, the available bandwidth, and any congestion in the network.
7. Optimize the routing algorithm to minimize the delay, packet loss, and congestion in the network. This can be achieved by using heuristics such as load balancing, path redundancy, and traffic engineering.
8. Test the network with various traffic patterns and analyze its performance. This can help identify any bottlenecks, congestion, or routing issues that need to be addressed.

**Finite State Machine**

Mesh Network

**\**

**State 1: Initialization**

LoRa modules on each node are initialized with appropriate settings such as frequency, spreading factor, bandwidth, and coding rate.

**State 2: Listen**

Each node listens for incoming messages and stores any incoming messages in a buffer.

**State 3: Direct Transmission**

When a node wants to send a message, it checks if the destination node is within its radio range. If it is, the message is sent directly to the destination node and the state machine returns to the Listen state.

**State 4: Broadcast**

If the destination node is not within radio range, the sender broadcasts the message to all nodes within its radio range and the state machine moves to the Receive state.

**State 5: Receive**

When a node receives a broadcast message, it checks if it is the destination node. If it is, it stores the message in its buffer and sends an acknowledgement message back to the sender, transitioning back to the Listen state.

**State 6: Duplicate Check**

If the node is not the destination node, it checks if it has already received the message before. If it has, it discards the message and returns to the Listen state.

**State 7: Forwarding**

If the node has not received the message before, it stores the message in its buffer and rebroadcasts it to all nodes within its radio range except for the sender. The state machine returns to the Listen state.

**State 8: Route Discovery**

If a node is unable to communicate with any of its neighbors, it broadcasts a message requesting help to find a new route to the destination node and enters the Route Discovery state.

**State 9: Route Response**

Nodes receiving a request for help message respond with a message containing the best route they know to the destination node. The requesting node selects the best route and starts using it to send messages to the destination node. The state machine returns to the Listen state.

**State 10: Update Routing Table**

Nodes periodically update their routing tables to reflect changes in the network topology, and the state machine stays in this state until the update is complete