**Flood Monitoring and Early Warning System.**

**Phase-3**

**Executive Summary:**

This document outlines a comprehensive strategy to transform the initial design for flood monitoring and early warning into a practical and innovative solution. By following a structured process, we aim to improve the safety, reliability, and overall wellbeing of public for the residences leaving near flood-prone areas

**Introduction:**

The key concept of this project is to create an efficient and comprehensive flood monitoring and early warning system to protect flood-prone areas and to reduce the risks to residents and infrastructure. This document details the steps to transform the initial design into a ground breaking innovation.

**Tasks:**

* Deploy IoT sensors (eg. Ultrasonic ) in the water bodies to gather information about the water levels.
* Deploy a Python script on the IoT sensors to send the real-time water level and warnings.

**Code:**

Main:

import RPi.GPIO as GPIO

import dht11

import time

import datetime

# initialize GPIO

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

# read data using pin 14

instance = dht11.DHT11(pin=14)

GPIO.setup(16,GPIO.OUT)

def THRESHOLD():

return 30

class ultraSound:

TRIG = 20

ECHO = 21

#LED=16

def \_\_init\_\_(self):

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(self.TRIG,GPIO.OUT)

GPIO.setup(self.ECHO,GPIO.IN)

#GPIO.setup(self.LED,GPIO.OUT)

def readData(self):

GPIO.output(self.TRIG, False)

#print "Waitng For Sensor To Settle"

time.sleep(2)

GPIO.output(self.TRIG, True)

time.sleep(0.00001)

GPIO.output(self.TRIG, False)

pulse\_start=0;

pulse\_end=0;

while GPIO.input(self.ECHO)==0:

pulse\_start = time.time()

while GPIO.input(self.ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

time.sleep(0.4)

return distance;

sensor=ultraSound();

while True:

result = instance.read()

if result.is\_valid():

print("Last valid input: " + str(datetime.datetime.now()))

print("Temperature: %-3.1f C" % result.temperature)

print("Humidity: %-3.1f %%" % result.humidity)

time.sleep(3)

print"----------------------------------------------------"

sensor.readData();

if(sensor.readData()<=THRESHOLD()):

GPIO.output(16,GPIO.HIGH)

print "WARNING THE LEVEL OF WATER HAS RAISED BEYOND THE THRESHOLD LEVEL"

else:

GPIO.output(16,GPIO.LOW)

print"----------------------------------------------------"

DHT11 Sensor:

import time

import RPi

class DHT11Result:

'DHT11 sensor result returned by DHT11.read() method'

ERR\_NO\_ERROR = 0

ERR\_MISSING\_DATA = 1

ERR\_CRC = 2

error\_code = ERR\_NO\_ERROR

temperature = -1

humidity = -1

def \_\_init\_\_(self, error\_code, temperature, humidity):

self.error\_code = error\_code

self.temperature = temperature

self.humidity = humidity

def is\_valid(self):

return self.error\_code == DHT11Result.ERR\_NO\_ERROR

class DHT11:

'DHT11 sensor reader class for Raspberry'

\_\_pin = 0

def \_\_init\_\_(self, pin):

self.\_\_pin = pin

def read(self):

RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.OUT)

# send initial high

self.\_\_send\_and\_sleep(RPi.GPIO.HIGH, 0.05)

# pull down to low

self.\_\_send\_and\_sleep(RPi.GPIO.LOW, 0.02)

# change to input using pull up

RPi.GPIO.setup(self.\_\_pin, RPi.GPIO.IN, RPi.GPIO.PUD\_UP)

# collect data into an array

data = self.\_\_collect\_input()

# parse lengths of all data pull up periods

pull\_up\_lengths = self.\_\_parse\_data\_pull\_up\_lengths(data)

# if bit count mismatch, return error (4 byte data + 1 byte checksum)

if len(pull\_up\_lengths) != 40:

return DHT11Result(DHT11Result.ERR\_MISSING\_DATA, 0, 0)

# calculate bits from lengths of the pull up periods

bits = self.\_\_calculate\_bits(pull\_up\_lengths)

# we have the bits, calculate bytes

the\_bytes = self.\_\_bits\_to\_bytes(bits)

# calculate checksum and check

checksum = self.\_\_calculate\_checksum(the\_bytes)

if the\_bytes[4] != checksum:

return DHT11Result(DHT11Result.ERR\_CRC, 0, 0)

# ok, we have valid data

# The meaning of the return sensor values

# the\_bytes[0]: humidity int

# the\_bytes[1]: humidity decimal

# the\_bytes[2]: temperature int

# the\_bytes[3]: temperature decimal

temperature = the\_bytes[2] + float(the\_bytes[3]) / 10

humidity = the\_bytes[0] + float(the\_bytes[1]) / 10

return DHT11Result(DHT11Result.ERR\_NO\_ERROR, temperature, humidity)

def \_\_send\_and\_sleep(self, output, sleep):

RPi.GPIO.output(self.\_\_pin, output)

time.sleep(sleep)

def \_\_collect\_input(self):

# collect the data while unchanged found

unchanged\_count = 0

# this is used to determine where is the end of the data

max\_unchanged\_count = 100

last = -1

data = []

while True:

current = RPi.GPIO.input(self.\_\_pin)

data.append(current)

if last != current:

unchanged\_count = 0

last = current

else:

unchanged\_count += 1

if unchanged\_count > max\_unchanged\_count:

break

return data

def \_\_parse\_data\_pull\_up\_lengths(self, data):

STATE\_INIT\_PULL\_DOWN = 1

STATE\_INIT\_PULL\_UP = 2

STATE\_DATA\_FIRST\_PULL\_DOWN = 3

STATE\_DATA\_PULL\_UP = 4

STATE\_DATA\_PULL\_DOWN = 5

state = STATE\_INIT\_PULL\_DOWN

lengths = [] # will contain the lengths of data pull up periods

current\_length = 0 # will contain the length of the previous period

for i in range(len(data)):

current = data[i]

current\_length += 1

if state == STATE\_INIT\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# ok, we got the initial pull down

state = STATE\_INIT\_PULL\_UP

continue

else:

continue

if state == STATE\_INIT\_PULL\_UP:

if current == RPi.GPIO.HIGH:

# ok, we got the initial pull up

state = STATE\_DATA\_FIRST\_PULL\_DOWN

continue

else:

continue

if state == STATE\_DATA\_FIRST\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# we have the initial pull down, the next will be the data pull up

state = STATE\_DATA\_PULL\_UP

continue

else:

continue

if state == STATE\_DATA\_PULL\_UP:

if current == RPi.GPIO.HIGH:

# data pulled up, the length of this pull up will determine whether it is 0 or 1

current\_length = 0

state = STATE\_DATA\_PULL\_DOWN

continue

else:

continue

if state == STATE\_DATA\_PULL\_DOWN:

if current == RPi.GPIO.LOW:

# pulled down, we store the length of the previous pull up period

lengths.append(current\_length)

state = STATE\_DATA\_PULL\_UP

continue

else:

continue

return lengths

def \_\_calculate\_bits(self, pull\_up\_lengths):

# find shortest and longest period

shortest\_pull\_up = 1000

longest\_pull\_up = 0

for i in range(0, len(pull\_up\_lengths)):

length = pull\_up\_lengths[i]

if length < shortest\_pull\_up:

shortest\_pull\_up = length

if length > longest\_pull\_up:

longest\_pull\_up = length

# use the halfway to determine whether the period it is long or short

halfway = shortest\_pull\_up + (longest\_pull\_up - shortest\_pull\_up) / 2

bits = []

for i in range(0, len(pull\_up\_lengths)):

bit = False

if pull\_up\_lengths[i] > halfway:

bit = True

bits.append(bit)

return bits

def \_\_bits\_to\_bytes(self, bits):

the\_bytes = []

byte = 0

for i in range(0, len(bits)):

byte = byte << 1

if (bits[i]):

byte = byte | 1

else:

byte = byte | 0

if ((i + 1) % 8 == 0):

the\_bytes.append(byte)

byte = 0

return the\_bytes

def \_\_calculate\_checksum(self, the\_bytes):

return the\_bytes[0] + the\_bytes[1] + the\_bytes[2] + the\_bytes[3] & 255