IOT based smart Farming using Field Programmable Gate Array

PROJECT

Advanced Digital System Design with FPGAs

(EEE4019)

FACULTY – PROF. BALAMURUGAN S
SLOT - E2

Moiz Kharodawala, 19BEE0165
B.Tech., Electrical and Electronics, VIT, Vellore, India.

moiz.shabbir2019@vitstudent.ac.in

Anmol Khantaal,19BEE0153
B.Tech., Electrical and Electronics, VIT, Vellore, India.

anmol.khantaal2019@vitstudent.ac.in

***Abstract*— Agriculture plays a vital role in the development of a country. In India about 70% of the population depends upon farming and one-third of the nation's capital comes from farming. Issues concerning agriculture have been always hindering the development of the country. Internet of Things (IoT) technology has brought revolution to each and every field of common man's life by making everything smart and intelligent. The development of Intelligent Smart Farming IoT based devices is day by day turning the face of agriculture production by not only enhancing it but also making it cost-effective and reducing wastage. The objective of this project was to develop a smart sensor-based monitoring system for agricultural environment using field programmable gate array (FPGA) which comprised of wireless protocol, different types of sensors, microcontroller, serial protocol and the field programmable gate array with display element. Different types of sensors such as temperature, soil moisture and relative humidity senses the data in agricultural environment and provide it to microcontroller. Data are displayed on LCD unit to monitor the system. Relays are interfaced with FPGA acting as a controlling element according to conditions provided to it.**

Index Terms*— Agriculture, FPGA, microcontroller, wireless sensor network.

I. INTRODUCTION

Our system consists of two main parts, one part monitors the environmental parameters such as temperature, humidity and soil moisture whereas the other part deals with the irrigation process. A battery level monitoring system monitors the environment with the help of various sensors. This helps in analysing various parameters affecting the yield of crops.

An irrigation system takes care of watering the crop based on the readings taken from various sensors. This system helps in reducing the wastage of water used in irrigation on a day-to-day basis. The project monitors the farm or greenhouse and based upon the readings of different kinds of sensors like temperature, humidity, soil moisture, and displays types of messages on the serial monitor about the present conditions so that the farmer can take quick action. This will increase the quantity and quality of the crops by properly monitoring the various present conditions.

Agriculture has been one of the primary occupations of man since early civilizations and even today manual interventions in farming are inevitable. There are many plants that are very sensitive to water levels and required specific level of water supply for proper growth, if this not they may die or results in improper growth. It's hardly possible that every farmer must possess the perfect knowledge about growing specifications of plants in case of water supply. So, to have a help, an attempt is made by introducing the proposed project named Smart Sensors Based Monitoring System for Agriculture Using FPGA. Water saving, improvement in agricultural yield and automation in agriculture are the objectives of this project. By the use of sensors in project, awareness about changing conditions of moisture, temperature and humidity level will be made available for the farmers so that according to changing conditions of moisture, temperature and humidity farmers will be able to schedule the proper timing for water supply and all the necessary things that required for proper growth of plants. This system can also be used in greenhouses to control important parameters like temperature, soil moisture, humidity etc as per the requirement of proper growth of plants.

II. REVIEW OF AGRICULTURAL BASED SYSTEMS

After the researches, it was found that the yield of agriculture goes on decreasing day by day. Use of technology in the field of agriculture plays important role in increasing the production as well as in reducing the extra man power efforts.
Some of the researchers tried for betterment of farmers and provided the systems that use technologies which are helpful for increasing the agricultural yield. In one of the researches, system was developed using sensor network for precision irrigation and the system focuses on evapotranspiration and soil moisture mainly [1]. Y. Kim et al. designed system related with the distributed wireless sensor network useful for remote sensing and controlling the irrigation system. The objective of the system was to maximize the productivity with minimal use of water [2]. In another research related with agricultural thing, researcher designed the system for measurement of soil parameters using wireless sensor network in such way that the sensors were placed under the soil measuring the soil related parameters. I. Aziz et al. research focused on developing a system that can remotely monitor and predict changes in temperature level in agricultural greenhouse [3].

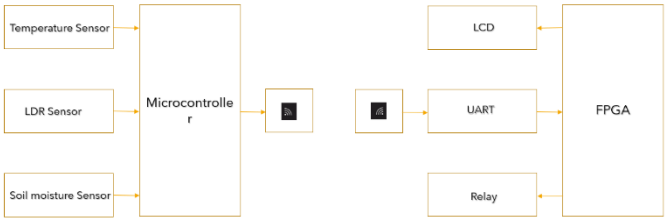
III. BRIEF DESCRIPTION OF THE DEVELOPED SYSTEM

In recent years, the agricultural industry has become more cognizant of the importance of integrating technology to improve the environment. Manual data collection for desired factors can be occasional, not continuous, and result in fluctuations due to inaccurate measurement. This can make it

difficult to control critical environmental elements. The use of wireless separate sensor nodes can cut down on the time and effort needed to monitor an area. Data logging reduces the likelihood of data being lost or misplaced. It would also allow personnel to be placed in important locations without putting them in dangerous situations. Quicker response times to adverse causes and situations, greater quality control of the output, and lower labour costs are all advantages of monitoring systems. Temperature, humidity, atmospheric pressure, and other variables could be measured remotely using technology.

Receiver The system comprises of sensors, microcontroller, wireless Bluetooth modules, FPGA board and LCD as a displaying element. Sensors sense the data and provide it to the microcontroller which is connected to the wireless module. The data is transmitted wirelessly and is provided to the FPGA where thresholds for the sensors can be varied according to need and decides whether the relays to be get ON/OFF. Simultaneously the sensors readings will be displayed on LCD. FPGA can be programmed to perform the needed actions by employing relays like turning on or off of water pump or sprayer. The system has an LCD display for continuously alerting the user about the condition in the field, the entire setup becomes user friendly.

Fig. 1. Proposed sensors based agricultural system



IV. DISSCRIPTION DEVELOPED AGRICULTURAL SYSTEM

The proposed system comprised of sensing unit, monitoring unit, auxiliary unit and wireless unit. The sensing unit made up of different types of sensors such as temperature, soil moisture, and relative humidity sensors. Monitoring unit consists of the microcontroller and FPGA element. The auxiliary unit includes LCD and relay element whereas wireless unit comprises of Bluetooth module both at the transmitter and receiver side. The system is described in details as follows.

A. Temperature Sensor

You can measure temperature more accurately than using a thermistor. The sensor circuitry is sealed and not subject to oxidation, etc. The DHT11 generates a higher output voltage than thermocouples and may not require that the output voltage be amplified

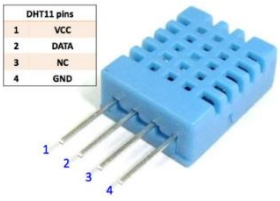


Fig. 2. LM 35 temperature sensor

B. Soil Moisture Sensor

Moisture is nothing but water contents in the soil. India is a country where we find the lot of difference in the quality of soil. This difference in quality of soil differ the water contents in the soil ultimately the moisture. So it is very necessary to study such parameter which affects the growth of plants and ultimately the yield. The monitoring of such parameter can be done using some sensor which gives the accurate and precise knowledge about the water contents. For that the sensor was designed using a capacitive bridge and resistors measuring moisture level of soil shown in Figure 3. Here, two aluminium foils were used which acts as capacitive plates and soil acts as dielectric medium. As the water level or moisture level increases, it will increase the electrical conductivity of medium thereby reducing the voltage across the plates.

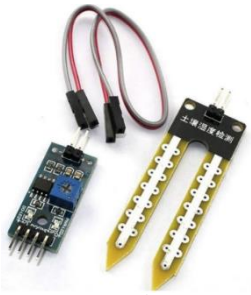


Fig. 3. Circuit Diagram of Soil Moisture Sensor

C. Wireless Module Nodemcu

NodeMCU is an open source firmware for which open source prototyping board designs are available. The name "NodeMCU" combines "node" and "MCU" (micro-controller unit). The term "NodeMCU" strictly speaking refers to the firmware rather than the associated development kits. Both the firmware and prototyping board designs are open source. The firmware uses the Lua scripting language. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented.

D.FIELD PROGRAMMABLE GETE ARREY (FPGA)

A Field-Programmable Gate Array is an integrated circuit designed to be configured by a customer or a designer after manufacturing, hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs relative to an ASIC design offers advantages for many applications. In most FPGAs, the logic blocks also include memory elements, which may be

simple flip-flops or more complete blocks of memory. In addition to digital functions, some FPGAs have analog features. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signalling channels. A few "mixed signal FPGAs" have integrated peripheral analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip. Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric [13].

G. Liquid Crystal Display (LCD)

A liquid crystal display is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. Each pixel consists of a column of liquid crystal molecules suspended between two transparent electrodes, and two polarizing filters, the axes of polarity of which are perpendicular to each other. Without the liquid crystals between them, light passing through one would be blocked by the other. In the developed system, a display of 16x2 is present on the FPGA board. The on-board LCD was used to display the values read by different sensors. The interfacing of LCD with the system was done through the FSM modelling in Active HDL as shown in Figure 7.

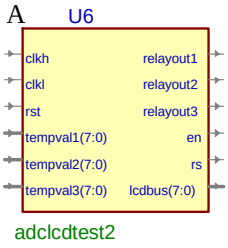


Fig. 7. LCD Block for Displaying Sensors Parameter

LCD plays an important role in displaying the parameters making the system user friendly. It displays the three different values of sensors that has sensed by the sensors through environmental condition.

H. FPGA Implementated Agricultural System

We are objecting to control temperature and water of an expensive plant. We just want to make sure that it won't die :). To do that, we are using water (humidity) and temperature sensors. We are assuming that outputs of water and temperature sensors are four and three bits. These output bits are inputs of the circuit. Sensors take analog information and sample it to a digital signal.

Sample coding for water and temperature sensors are:

Water (Humidity) Sensor Output Legend: 0%: 0(0000); 10%: 1(0001); 20%: 2(0010);30%: 3(0011); 40%: 4(0100); 50%: 5(0101); 60%: 6(0110); 70%: 7(0111); 80%: 13(1101); 90%: 14(1110); 100%: 15(1111)

Temperature Sensor Output Legend (in Celsius): <15: 0(000); 17.5: 1(001); 20: 2(010); 22.5: 3(011); 25: 4(100); 27.5: 5(101); 30: 6(110); 32.5: 7(111);

We have four inputs called **start**, **reset**, **water**, and **temperature**:

- input **start** controls the POWER, either POWER is ON or OFF
- input **reset** puts the system into the default state.
- input **water** is the output of the humidity sensor
- input **temperature** is the output of the temperature sensor

We have two outputs called **pump** and **ac**:

- output **pump** (water pump): ON (1) or OFF (0)
- output **ac** (Air Conditioning): A/C: Heating (11), Cooling (01) or OFF (00)

We are assuming that the pump only drips and heating is more dominant than cooling.

We have seven states:

- **SO**: Power is OFF (both pump and ac are closed)
- **S1**: Default state (pump is ON and ac is OFF)
- **S2**: Default state (ac is OFF)
- **S3**: Default state (ac is Heating)
- **S4**: Default state (ac is Cooling)
- **S5**: Default state (pump is ON)
- **S6**: Default state (pump is OFF)

Since **S2-S4** and **S5-S6** states are independent. We register two state variable state_water and state_temperature

operating conditions were provided while implementing the data inputs from the sensor to the FPGA element.

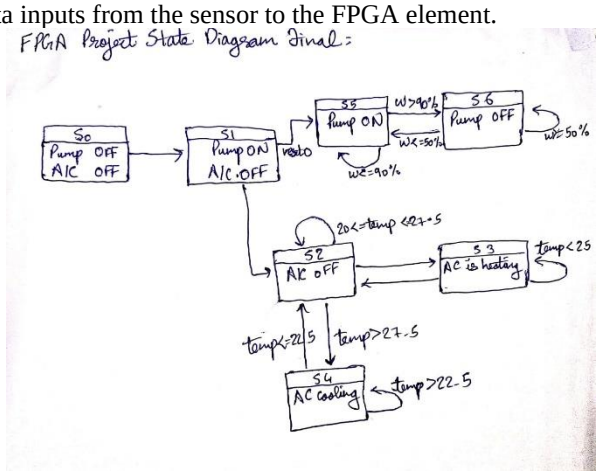


Fig. 8. FPGA FSM for the system of Agriculture

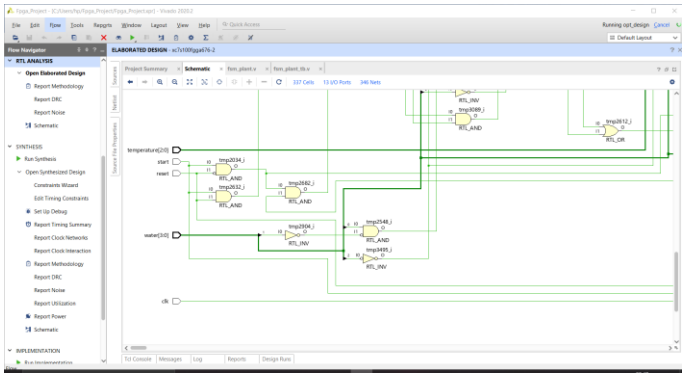


Fig. 9. FPGA Implemented System for Agriculture

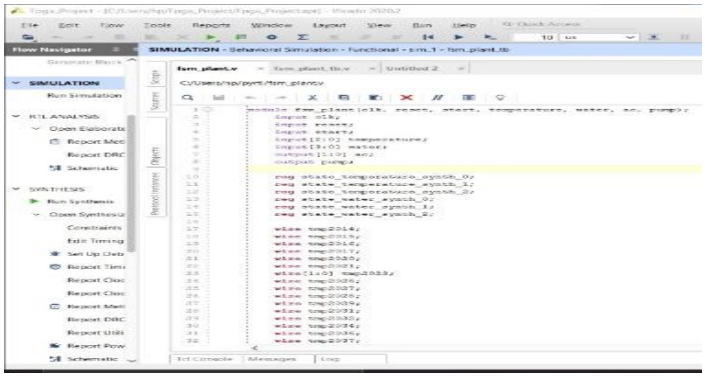
V. Codes

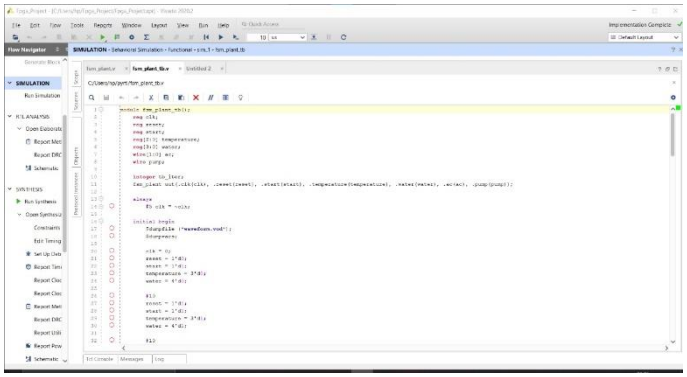
A. *Python Code to generate FSM*

```
# # Finite State Machine for Water and Temperature Automation of a Plant

# In our desgin we are using water (humidity) and temperature sensors. Output of water and temperature sensors are four and three bits. These output bits are inputs of circuit. Sensors take analog information and samples it to a digital signal.
#
# Sample coding for water and temperature sensors are:
#
# **Water (Humidity) Sensor Output Legend:**
# * 0%: 0(0000); 10%: 1(0001); 20%: 2(0010);30%: 3(0011); 40%: 4(0100); 50%: 5(0101);
# * 60%: 6(0110); 70%: 7(0111); 80%: 13(1101); 90%: 14(1110); 100%: 15(1111)
#
# **Temperature Sensor Output Legend (in Celsius):**
# * <15: 0(000); 17.5: 1(001); 20: 2(010); 22.5: 3(011); 25: 4(100); 27.5: 5(101); 30: 6(110); 32.5: 7(111);
#
# We have four inputs called **start**, **reset**, **water**, and **temperature**
#
# * input **start** controls the POWER, either POWER is ON or OFF
# * input **reset**  puts the system into default state.
# * input **water** is the output of the humidity sensor
# * input **temperature** is the output of the the temperature sensor
#
# We have two outputs called **pump** and **ac**,
#
# * output **pump** (water pump): ON (1) or OFF (0)
#
# * output **ac** (Air Conditioning): A/C: Heating (11), Cooling (01) or OFF (00)
#
# We have seven states. For State machine diagram please visit:
http://venividiwiki.ee.virginia.edu/mediawiki/index.php/Water_and_Temperature_Automation_of_a_Plant
# or
http://venividiwiki.ee.virginia.edu/mediawiki/index.php/Water_and_Temperature_Automation_of_a_Plant_using_FPGA
#
# * **SO**: Power is OFF (both pump and ac are closed)
# * **S1**: Default state (**pump** is ON and **ac** is OFF)
# * **S2**: Default state (**ac** is OFF)
# * **S3**: Default state (**ac** is Heating)
# * **S4**: Default state (**ac** is Cooling)
# * **S5**: Default state (**pump** is ON)
# * **S6**: Default state (**pump** is OFF)
#
# Since **S2-S4** and **S5-S6** states are independent. We register two state variable **state_water** and **state_temperature**

# # Inport the pyrtl library

import pyrtl
pyrtl.reset_working_block()


# ## Define wires and registes
# **Input** and **Output** are special type of wire, for hardware match. We need to use **Register** for states to give the decision with ".next"

start = pyrtl.Input(1, 'start')
reset = pyrtl.Input(1, 'reset')
water = pyrtl.Input(4, 'water')
temperature = pyrtl.Input(3, 'temperature')
pump = pyrtl.Output(1, 'pump')
ac = pyrtl.Output(2, 'ac')
state_water = pyrtl.Register(3, 'state_water') #S0 S1 S5 S6
state_temperature = pyrtl.Register(3, 'state_temperature') #S0 S1 S2 S3 S4


# First new step, let's **enumerate a set of constant to serve as our states**


S0, S1, S2, S3, S4, S5, S6 = [pyrtl.Const(x, bitwidth=3) for x in range(7)]


# ## Create state transition and logic
# **PyRTL has a class "ConditionalUpdate"** to provide a predicated update to a registers, wires, and memories.
# It is same as "if-else if-else" conditional states.

with pyrtl.conditional_assignment:
    with start:  # power is ON
        with reset:  # system in reset
            state_water.next |= S1
            state_temperature.next |= S1
        with pyrtl.otherwise: # functioning
            with (state_water == S0): # if previous state_water is power OFF
                state_water.next |= S1
                with (state_temperature == S0):
                    state_temperature.next |= S1
            with (state_water == S1):
                state_water.next |= S5
                with (state_temperature == S1):
                    state_temperature.next |= S2
            with state_water == S5: # pump ON
                with water <= 14: # less than or equal to %90
                    state_water.next |= S5
                    with state_temperature == S2: #ac OFF
                        with temperature < 2: # less than 20
                            state_temperature.next |= S3
                        with temperature > 5: # higher than 27.5
                            state_temperature.next |= S4
                        with pyrtl.otherwise: # between 20 and 27.5
                            state_temperature.next |= S2
                    with state_temperature == S3: # heat untill 25
                        with temperature < 4: # less than 25
                            state_temperature.next |= S3
                        with temperature >= 4: # higher than or equal to 25
                            state_temperature.next |= S2
                    with state_temperature == S4: #cool untill 22.5
                        with temperature > 3: # higher than 22.5
                            state_temperature.next |= S4
```

```
                with temperature <= 3: # less than or equal to 22.5
                    state_temperature.next |= S2
            with water > 14: # higher than %90
                state_water.next |= S6
                with state_temperature == S2: #ac OFF
                    with temperature < 2: # less than 20
                        state_temperature.next |= S3
                    with temperature > 5: # higher than 27.5
                        state_temperature.next |= S4
                    with pyrtl.otherwise: # between 20 and 27.5
                        state_temperature.next |= S2
                with state_temperature == S3: # heat untill 25
                    with temperature < 4: # less than 25
                        state_temperature.next |= S3
                    with temperature >= 4: # higher than or equal to 25
                        state_temperature.next |= S2
                with state_temperature == S4: #cool untill 22.5
                    with temperature > 3: # higher than 22.5
                        state_temperature.next |= S4
                    with temperature <= 3: # less than or equal to 22.5
                        state_temperature.next |= S2
        with state_water == S6: # pump OFF
            with water <= 5: # less than or equal to %50
                state_water.next |= S5
                with state_temperature == S2: #ac OFF
                    with temperature < 2: # less than 20
                        state_temperature.next |= S3
                    with temperature > 5: # higher than 27.5
                        state_temperature.next |= S4
                    with pyrtl.otherwise: # between 20 and 27.5
                        state_temperature.next |= S2
                with state_temperature == S3: # heat untill 25
                    with temperature < 4: # less than 25
                        state_temperature.next |= S3
                    with temperature >= 4: # higher than or equal to 25
                        state_temperature.next |= S2
                with state_temperature == S4: #cool untill 22.5
                    with temperature > 3: # higher than 22.5
                        state_temperature.next |= S4
                    with temperature <= 3: # less than or equal to 22.5
                        state_temperature.next |= S2
            with water > 5: # higher than %50
                state_water.next |= S6
                with state_temperature == S2: #ac OFF
                    with temperature < 2: # less than 20
                        state_temperature.next |= S3
                    with temperature > 5: # higher than 27.5
                        state_temperature.next |= S4
                    with pyrtl.otherwise: # between 20 and 27.5
                        state_temperature.next |= S2
                with state_temperature == S3: # heat untill 25
                    with temperature < 4: # less than 25
                        state_temperature.next |= S3
                    with temperature >= 4: # higher than or equal to 25
                        state_temperature.next |= S2
                with state_temperature == S4: #cool untill 22.5
                    with temperature > 3: # higher than 22.5
                        state_temperature.next |= S4
                    with temperature <= 3: # less than or equal to 22.5
                        state_temperature.next |= S2
        with pyrtl.otherwise:
            state_water.next |= S1
            state_temperature.next |= S1
    with pyrtl.otherwise: # power OFF
        state_water.next |= S0
        state_temperature.next |= S0


# **Stae Logic**

pump <<= (state_water == S1) | (state_water == S5)
ac <<= 3*(state_temperature == S3) + (state_temperature == S4) # for S3 (heating):
ac=11; for S4 (cooling): ac=01


# Now let's **print, build and test our state machine**.

print('--- plant_FSM Implementation ---')
print(pyrtl.working_block())
print()

sim_trace = pyrtl.SimulationTrace()
sim = pyrtl.Simulation(tracer=sim_trace)


# ## Simulate the FSM

# Rather than just give some random inputs, let's specify inputs to observe
functionality. The sim.step method takes a dictionary mapping inputs to their
values.  We could just specify the input set directly as a dictionary but it gets pretty
ugly -- let's use some python to parse them up.

sim_inputs = {
    'start':   '11111111111111111111100',
    'reset':   '01000000000000000000000',
    'water':   ['0', '1', '2','3', '4', '5','6','7','7',
'13','14','14','15','15','14','13','7','6','5','4','3','3','3'],
    'temperature':   ['0', '1', '2','1', '2', '3','4','5','6',
'6','6','5','4','4','3','3','4','5','6','1','1','1','1']
    }

for cycle in range(len(sim_inputs['start'])):
    sim.step({w: int(v[cycle]) for w, v in sim_inputs.items()})

print("--- Simulation Results ---")
sim_trace.render_trace(trace_list=['start', 'reset', 'water', 'pump',
'temperature','ac','state_water', 'state_temperature'])


# Also, to make our input/output easy to reason about we **specified an order to
the traces**

# ## Create Verilog code

# Output to a Verilog file named 'fsm_plant.v'
with open('fsm_plant.v', 'w') as v_file:
    pyrtl.OutputToVerilog(v_file)

# Output of TestBench to a Verilog file named 'fsm_plant_tb.v'
with open('fsm_plant_tb.v', 'w') as tbfile:
    pyrtl.output_verilog_testbench(dest_file=tbfile, simulation_trace=sim_trace)

# Optimized Verilog
pyrtl.synthesize()
pyrtl.optimize()

# Optimized output to a Verilog file named 'fsm_plant_opt.v'
with open('fsm_plant_opt.v', 'w') as vfile:
    pyrtl.Output_To_Verilog(vfile)
```

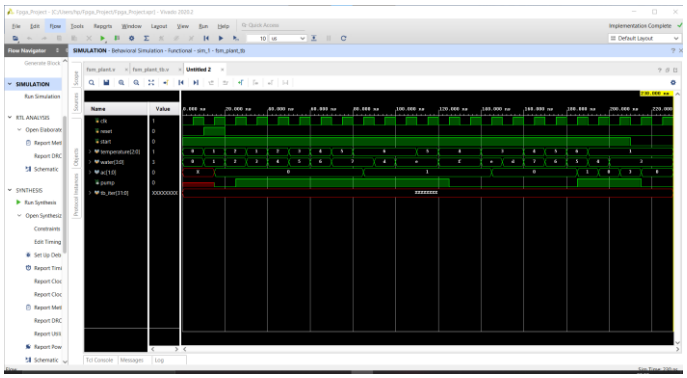
VI. SNAPSHOTS





The python script generates the following Verilog code and also links the various case outputs with the operations.

VII. RESULT AND ANALYSIS



VIII. CONCLUSION

In today's life, human beings are so busy that providing attention to work like manual irrigation, operating a water pump, insecticides and pesticides for the plants, *etc.*, is a typical task. So an automatic smart sensor based monitoring system is developed using FPGA which includes the environmental parameters such as temperature, soil moisture and relative humidity. The system was useful in monitoring the environmental conditions for any kind of plants thus providing the farmer necessary information to take precautions and prevent them from diseases that reduces the yield. Also transferring the data from sensing field to the destination field may have some obstacles in between, so to have proper transmission the given system is developed using the wireless protocol which is useful in transferring the data at a distance of about 25m with and without any kind of obstacles in between. Use of wireless protocol also made the system robust and the flexible one. The sensors that has used in the developed system are easy to relocate anywhere in the field as per the requirement and are précised too. The system is nothing but a step to increase yield of plants and to save water.

Using the proposed system fully automation can be possible in the field of agriculture and horticulture. The continuously decreasing costs of hardware and software, the wider acceptance of electronic systems in agriculture, and an emerging agricultural control system industry in several areas of agricultural production, will result in reliable control systems that will address several aspects of quality and quantity of production.

REFERENCES

[1] N. Shah and I. Das, "Precision Irrigation Sensor Network Based Irrigation," a book on Problems, Perspectives and Challenges of Agricultural Water Management, IIT Bombay, India, pp. 217–232, April 2008.

[2] Y. Kim, R. Evans and W. Iversen, "Remote Sensing and Control of an Irrigation System Using a Distributed Wireless Sensor Network," IEEE Transactions on Instrumentation and Measurement, , pp. 1379–1387 July 2008.

[3] Q. Wang, A. Terzis and A. Szalay, "A Novel Soil Measuring Wireless Sensor Network," IEEE Transactions on Instrumentation and Measurement, pp. 412–415, August 2010.

[4] I. Aziz, M. Hasan, M. Ismail, M. Mehat and N. Haron, "Remote Monitoring in Agricultural Greenhouse Using Wireless Sensor and Short Message Service," International Journal of Engineering Technology, Vol. 9, pp. 1–12, September 2010.

[5] Swarup S. Mathurkar, Rahul B. Lanjewar, Nilesh R. Patel, Rohit S. Somkuwar, Smart Sensors Based Monitoring System for Agriculture using Field Programmable Gate Array

[6] Yunseop (James) Kim, Robert G. Evans, and William M. Iversen Remote Sensing and Control of an Irrigation System Using a Distributed Wireless Sensor Network, ieee transactions on instrumentation and measurement, vol. 57, no. 7, july 2008

[7] Village precision poverty alleviation and smart agriculture based on FPGA and machine learning Lei Yin, Yu Zhang

[8] Muhammed Ihsan Husni, Mohammed Kareem Hussein, Mohd Shamian Bin Zainal, Shipun Anuar Bin Hamzah, Danial Bin Md Nor, Hazwaj Bin Mhd Poad, Soil Moisture Monitoring Using Field Programmable Gate Array, Indonesian Journal of Electrical Engineering and Computer Science Vol.11, No.1, July 2018, pp. 169~174

[9] Sridevi Navulur, A. S. C. S. Sastry, M. N. Giri Prasad, Agricultural Management through Wireless Sensors and Internet of Things, International Journal of Electrical and Computer Engineering (IJECE) Vol. 7, No. 6, December 2017, pp. 3492~3499

[10] Oukaira, A.; Benelhaouare, A.Z.; Kengne, E.; Lakhssassi, A. FPGA-Embedded Smart Monitoring System for Irrigation Decisions Based on Soil Moisture and Temperature Sensors. Agronomy 2021, 11, 1881. https://doi.org/10.3390/agronomy 11091881 Academic Editor: Dimitrios D. Alexakis